MOST User Guide

Politecnico di Torino

MARINE OFFSHORE RENEWABLE ENERGY LAB

Marine Offshore Renewable Energy Lab (MOREnergy Lab)
Department of Mechanical and Aerospace Engineering
Politecnico di Torino, Turin, Italy
November 2024

1 Introduction

MOST (Matlab for OFWT Simulation Tool), is a simulation software for simulating floating offshore wind turbines, hybrid wind-wave energy converters, platform with multiple turbines and actively controlled water ballast. It operates using WEC-Sim, an open-source software for the simulation of wave energy converters, developed in MATLAB/SIMULINK using the Simscape Multibody dynamics solver. An advantage of MOST is the reduced computational time compared to established codes such as OpenFAST due to the simplification of the aerodynamic computation. Look-up tables of aerodynamic and mooring loads can be obtained during pre-processing, saving computational time during the simulation. MOST also includes the control of the wind turbine which can be chosen between the Baseline and the ROSCO controllers ([4], [1]).

MOST has the possibility to model offshore renewable devices made of hydrodynamic bodies, mooring, wind turbines, joint and constraints and power-take-off (PTO) systems. The main novelty of MOST is the development of a wind turbine body which can be coupled with the WEC-Sim library extending the possibility to model floating offshore wind turbines and hybrid systems. MOST requires several user inputs similar to WEC-Sim including hydrodynamic bodies requiring hydrodynamic coefficients from Boundary Element Method (BEM) software (e.g. Nemoh, WAMIT or Ansys Aqwa), mooring, constraints, PTOs and simulation input details. Differently to WEC-Sim, MOST also includes user inputs relative to the wind turbine, wind and mooring which will be explained in the next sections. Numerical code added to WEC-Sim is the followings:

- MOST Simulink library (MOST.slx), which includes the wind turbine blocks.
- New classes (windTurbineClass and windClass) with details of the wind turbine and the wind input.
- Pre-processing to compute look-up tables of aerodynamic and mooring loads, wind turbine, wind field and controller properties.

WEC-Sim source code has also been modified to include further option features relative to the new capabilities. The following code has been modified:

- **initializeWecSim**: it is modified to add functions relative to the wind, wind turbine and new mooring blocks. New functions are created in each relative class to read data prepared during the pre-processing by the user. Control parameters are also added to give user choice of which control logic will be used.
- **mooringClass**: it is now also possible to calculate mooring loads (static loads) using non-linear look-up tables computed during pre-processing and with system displacements as input. Compared to before, the mooringClass now also has a function by means of which look-up tables are loaded from a file whose name is the new property called "lookupTableFile". In addition, it is also possible to simulate the mooring system by means of a quasi-static model (the same one used for the creation of look-up tables) similar to the way MAP++ software does. In this case, it will be sufficient to define the characteristics of the moorings among the 'mooringClass' class properties.

- **postProcessWecSim + responseClass**: wind turbine results are post-processed and saved in the same WEC-Sim output structure. Examples of outputs are the timeseries of wind turbine power, rotor speed, blade pitch and nacelle acceleration.

The following diagram summarizes the workflow for the simulation of wave energy converters, which has now been updated to include the simulation of floating offshore wind turbines or hybrid devices. In grey are highlighted the portions of the code introduced with MOST, in green the portions executed by WEC-Sim codes, and in yellow what is carried out by external software. The new functions introduced with MOST will be described below following the workflow in Figure 1.

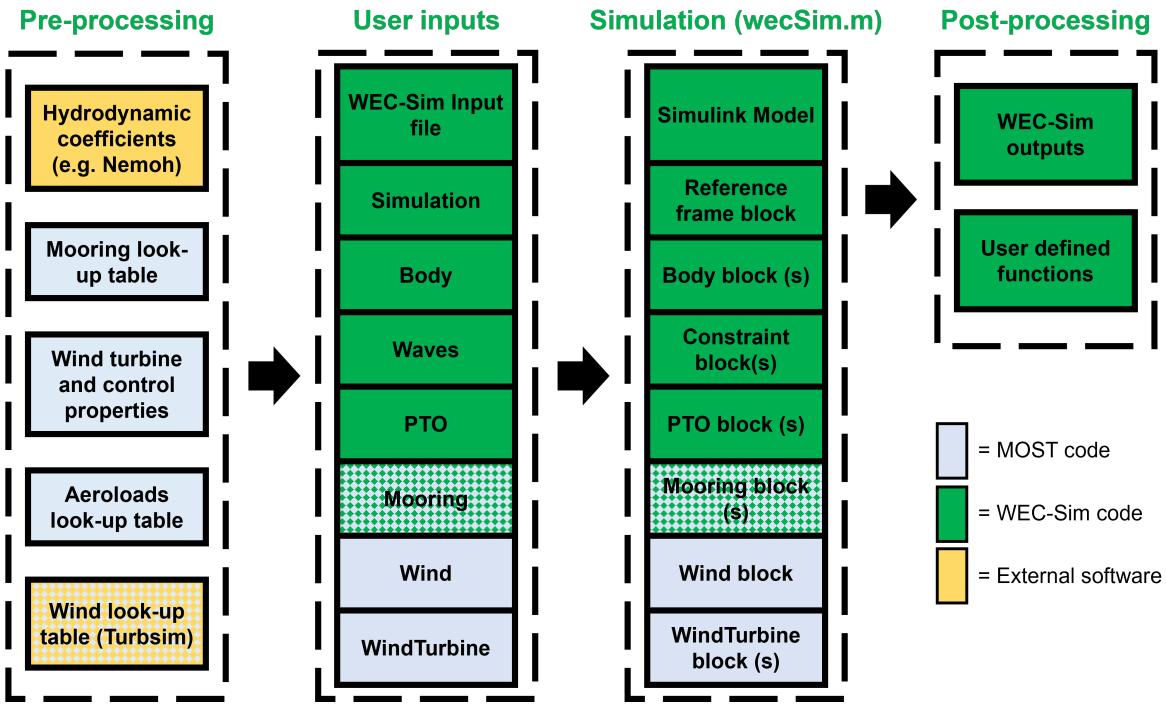

Figure 1: WEC-Sim workflow including MOST features

2 Getting Started

This section provides MATLAB requirements to use MOST and instructions on how to download WEC-Sim-MOST and how to run the examples.

2.1 MATLAB Requirements

WEC-Sim-MOST is developed in MATLAB/Simulink, and requires the following toolboxes:

Required Toolbox	Oldest Compatible Version
MATLAB	Version 9.9 (R2020b)
Simulink	Version 10.2 (R2020b)
Simscape	Version 5.0 (R2020b)
Simscape Multibody	Version 7.2 (R2020b)

Certain advanced features rely on external functions (such as MoorDyn), and additional MATLAB Toolboxes (such as Parallel Computing Toolbox and Optimization Toolbox).

Verify that the MATLAB required toolboxes are installed by typing `ver` in the MATLAB Command Window:

```
>> ver
-----------------------------------------------------------------------
MATLAB Version: 9.9.0.1592791 (R2020b)
-----------------------------------------------------------------------
MATLAB                                                Version 9.9         (R2020b)
Simulink                                              Version 10.2        (R2020b)
Simscape                                              Version 5.0         (R2020b)
Simscape Multibody                                    Version 7.2         (R2020b)
```

2.2 Download folders

To use MOST, the folders to download are WEC-Sim-MOST and MOST

2.3 Running examples

There are three sub-folders in the MOST/Examples folder relating to three different examples:

1. VolturnUS: Simulation of a system consisting of the floating platform of the semi-submersible type VolturnUS-S [2] on which the reference wind turbine IEA15MW [3] is mounted and with a catenary mooring system consisting of three lines.

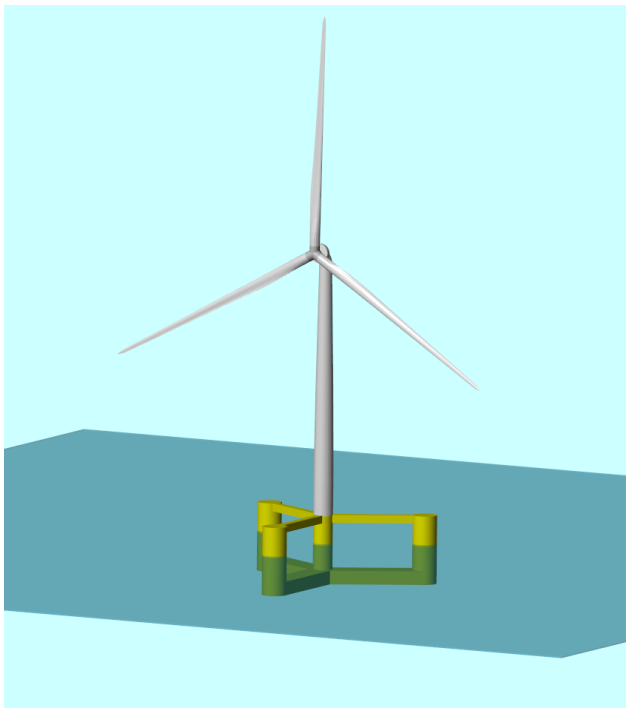

Figure 2: VolturnUS Example

2. 2Turbines: Simulation of two systems; the first is composed of a floating platform of the spar type on which is mounted the NREL5MW [6] wind turbine and with a catenary mooring system composed of three lines, while the second is composed of the floating platform of the semi-submersible type VolturnUS-S [2] on which is mounted the reference wind turbine IEA15MW [3] and with a catenary mooring system composed of three lines.

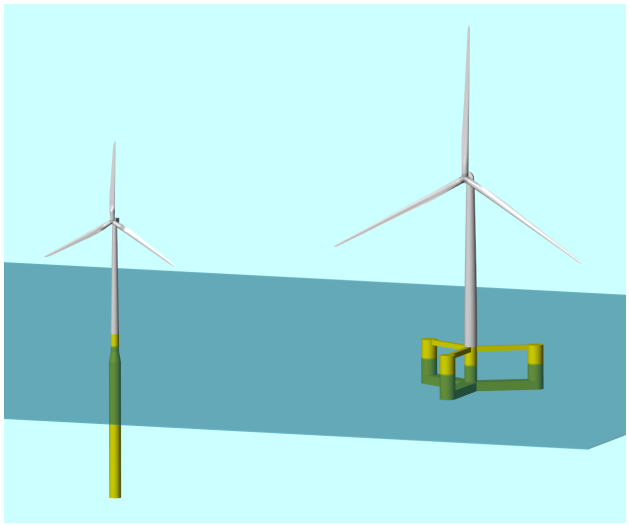

Figure 3: 2Turbines Example

3. HybridSpar: Simulation of a hybrid platform consisting of the IEA15MW [3] wind turbine mounted on a spar-type platform and a wave energy converter of the point-absorber type that draws energy from the relative translational motion with the platform.

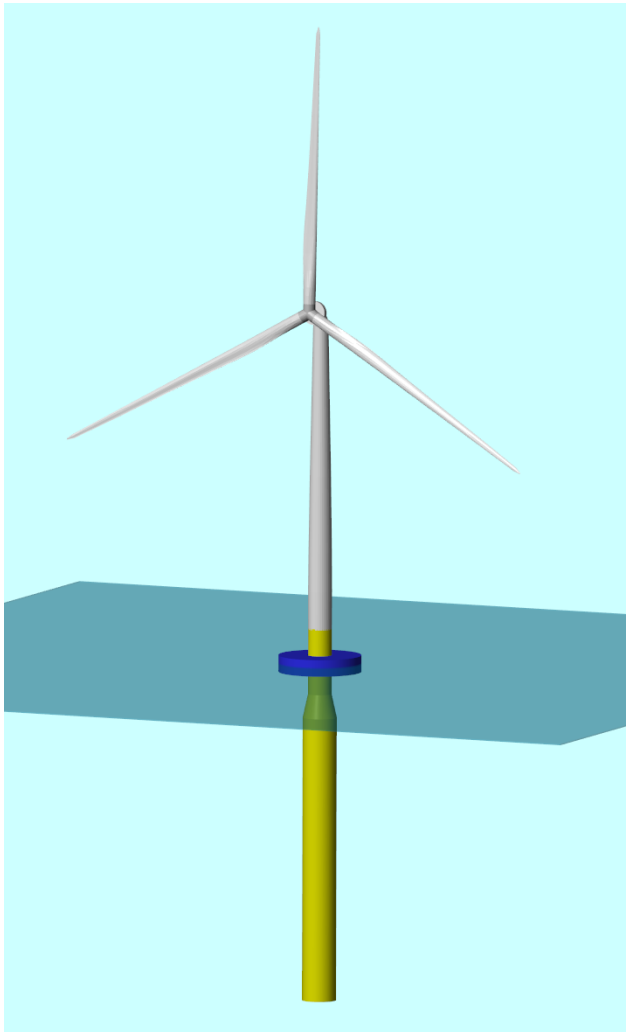

Figure 4: HybridSpar Example

To run an example, simply perform the following steps:

1. move to the 'WEC-Sim-MOST' folder and run the 'addWecSimSource.m' script. This will add the 'source' folder and its sub-folders to the MATLAB path.

2. move to the folder of one of the examples and run the script 'wecSim.m' by simply typing 'wecSim' in the command window.

It is also possible to change the simulation settings (such as simulation time, solver type or input wave type, etc.) by editing the script 'wecSimInputFile.m' in the same folder. Please consult the User Manual section for more information.

3 Theory

In this section, the main features introduced with MOST are illustrated and briefly explained from a theoretical point of view, in particular the wind field, the geometric and inertial characteristics of the wind turbine, the control logics to manage it, the way aerodynamic loads on the blades are calculated, and the mooring systems will be explained.

3.1 Wind

This section describes how the input wind field is generated; there are two possible methods: to have constant wind speed (in space) or to have a wind speed field in which turbulence is taken into account. It is possible to specify wind method in wecSimInputFile.m by choosing the value (0: turbulent, 1: constant) of the windClass class property "ConstantWindFlag". In the first case, it is possible to define a wind field that is the same for each point in the domain and that can vary over time. Specifically, it is possible to define a series of instants to which a certain wind speed and a certain azimuth and elevation angles correspond and the values of these at such instants. At intermediate times, there will be a linear interpolation between these characteristics. Regarding the second case, the scatter of the wind speed is obtained using an external code, Turbsim, developed by NREL, and integrated within the MOST code. The user can launch the RunTurbsim.m script to create the wind input data structure, specifying some properties such as mean velocity and turbulence intensity. For more information, it is recommended to read the "User Manual" section and the documentation of TurbSim [5]. The resulting data structure consists of the wind speed at each instant and for each node of a spatial grid covering the rotor area. During the simulation, the wind speed corresponding to the blade nodes will be obtained by interpolating between the grid points via look-up tables.

3.2 Wind turbine

3.2.1 Properties

All wind turbine's components are modeled as rigid bodies, they are the tower, the nacelle, the hub and the blades. The inertial and geometrical properties of the components must be defined in a MATLAB structure, the user can use the "WTproperties" scripts to write the parameters of the desired wind turbine. In particular, mass, moment of inertia, center of mass relative to the reference (see Figure 5) and center of mass in the global reference frame (whose origin is at still water level) are defined for each body. In addition, other parameters such as tilt and precone angles, tower height, electrical generator efficiency and CAD file names are set. The CAD files to define the geometry can be imported from external software. They must be saved in the folder "geometry". The user has to set the name of the CAD files in "WTcomponents" structure to allow MOST to upload the files.

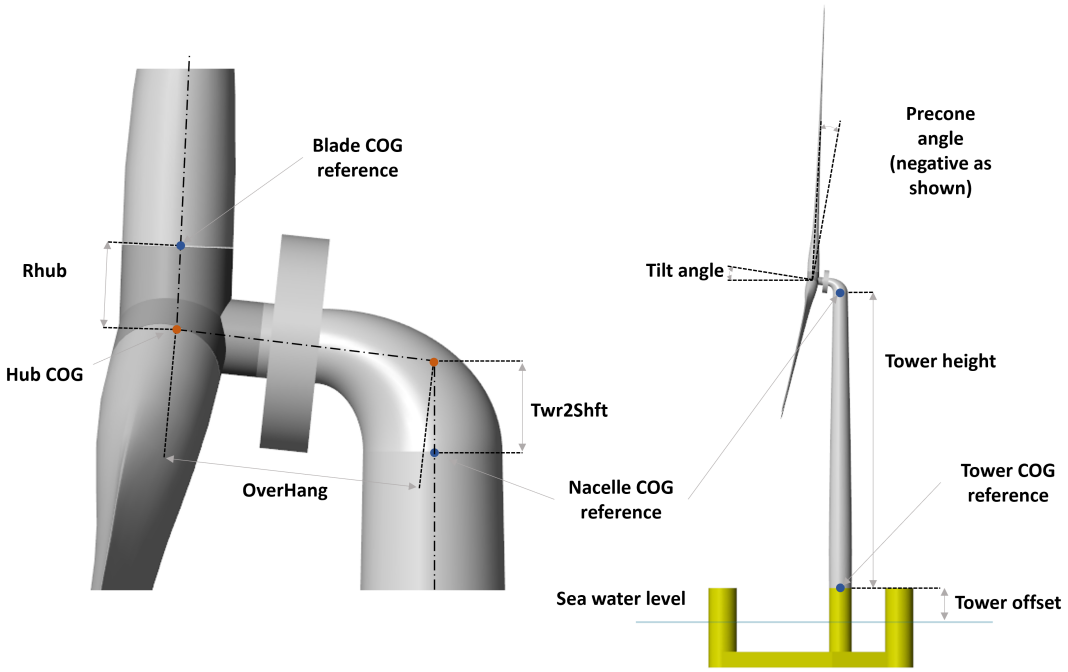

Figure 5: Wind turbine geometry

In addition to the general characteristics of the wind turbine, the user must set the specific properties for the blades by launching the "BladeData" scripts, which defines the needed data structure by taking the information from some text files in the "BladeData" subfolder. In these, lift, drag, and torque coefficients are specified for each type of airfoil used, as well as certain geometric characteristics of the blades such as twist angle and chord length as a function of radius and geometric characteristics related to pre-bending.

3.2.2 Control

In this section, we discuss how the wind turbine is controlled in MOST, two controls are present, the first is the control by which the extracted power is adjusted by acting on the braking torque due to the generator and the collective pitch angle of the blades, the second is the control that acts on the relative rotation between the tower and the RNA (Rotor Nacelle Assembly) which is responsible for maintaining perpendicularity between the main wind direction and the rotor area. The latter, known as yaw control, is a simple PID controller whose output is the relative rotation speed between the tower and RNA, and therefore does not conform to the actual controllers that deal with this implementation. However, it was decided to implement a simplified controller to allow interested users to include advanced control logic in a simple manner. In the Simulink model, there is in fact a subsystem dedicated to yaw control that can be modified according to the user's needs.

As far as the control acting on generator torque and blade angle is concerned, two different types of state-of-the-art controllers are implemented, the Baseline [4] and the ROSCO [1]) controllers. As for these, for the creation of the data required for the simulation, it is neces-

sary to know the steady-state values, i.e., the stationary values of certain quantities of interest when varying, in this case, the wind speed, which is considered constant for this purpose. The first step in obtaining the data required for the simulation is therefore to run the script called Steady_States.m in the sub-folder "mostData/windTurbine/control" to perform this calculation. Specifically, this calculates the stationary values of power, rotor speed, thrust force, generator torque, and blade pitch angle for both of the aforementioned control logic.

Only the ROSCO controller imposes an upper limit for the thrust force, so when the wind speed is close to the nominal wind speed (where the force peak occurs), the blade pitch value will be slightly higher to reduce the thrust and comply with the imposed limits. In the Baseline controller, no minimum rotor speed is imposed, which is the case for some turbine types in the ROSCO case.

Below is a figure 6 representing an example of steady-state values for Baseline and ROSCO controllers for the IEA 15 MW reference wind turbine [3].

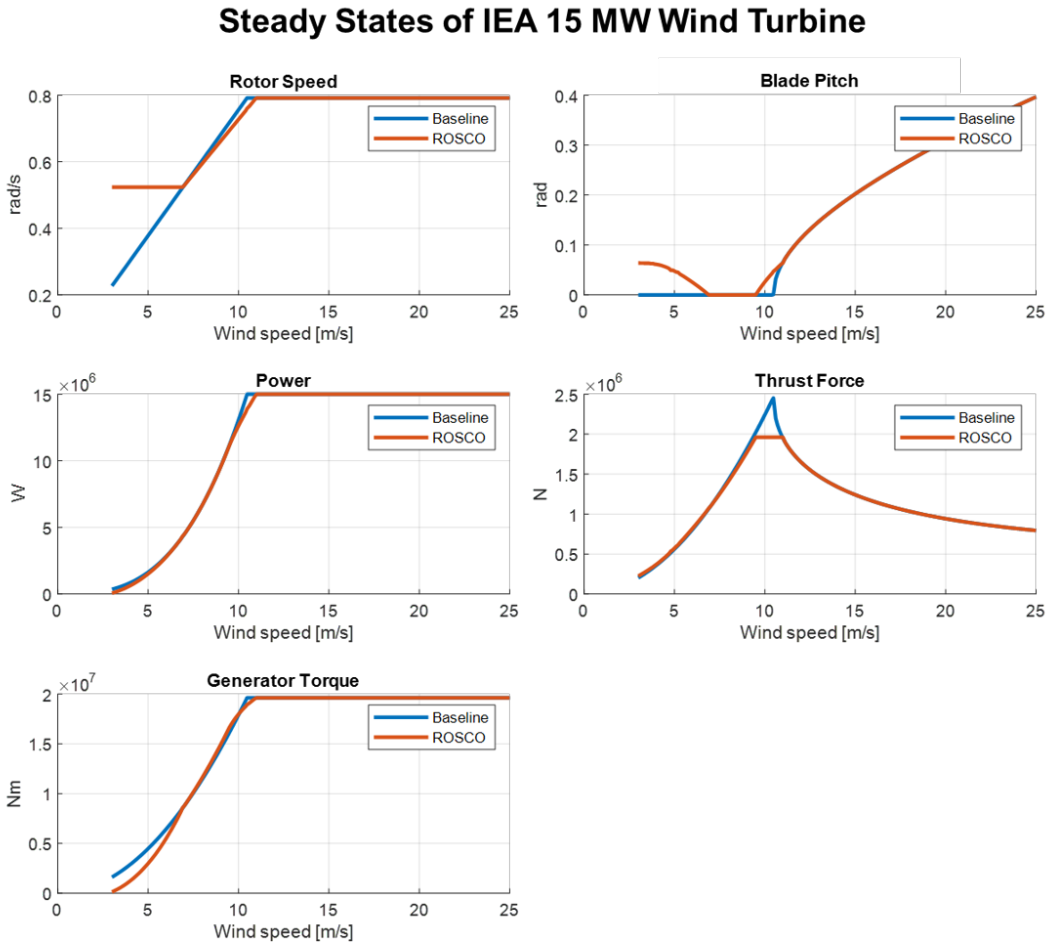

Figure 6: Example of steady-state values for Baseline and ROSCO controllers

In the following, the Baseline and ROSCO control logic will be briefly explained; for more information refer to [4] (Baseline) and [1] (ROSCO).

Baseline

Baseline is a conventional, variable-speed, variable collective pitch controller, which is made up of two independent systems:

- A generator torque controller designed to maximize power extraction below nominal wind speed.
- A blades collective pitch controller designed to regulate rotor and generator speed above nominal wind speed.

Generator Torque Controller

The generator-torque control law is designed to have three main regions and two transition ones between them. The generator torque is computed as a tabulated function of the filtered generator speed, incorporating 4 operational control regions: 1, 1.5, 2, and 3.

- **Region 1**: control region before cut-in wind speed, where the generator is detached from the rotor to allow the wind to accelerate the rotor for start-up. In this region, the generator torque is zero and no power is extracted from the wind.
- **Region 1.5**: transition region called start-up region and permits a smooth transition between null and optimal torque.
- **Region 2**: control region where extracted power is maximized. Here, to maintain the tip speed ratio constant at its optimal value, the generator torque is proportional to the square of the filtered generator speed. Aerodynamic torque can be expressed as:

$$T_{\text{aero}} = \frac{1}{2}\rho\pi\frac{R^5}{\lambda^3}C_P(\lambda,\theta_{\text{bl}})\cdot\Omega^2 = k_{\text{opt}}\cdot\Omega^2 \tag{1}$$

  Where k_{opt} is obtained with TSR (λ) and blade pitch values that lead to maximum power coefficient.
- **Region 3**: above rated condition region, where the generator torque is kept constant at its rated value. In this region pitch control is active to maintain rotor speed at its rated value.

The figure 7 below shows an example of the control law of the Baseline generator torque controller for the IEA 15 MW reference wind turbine [3].

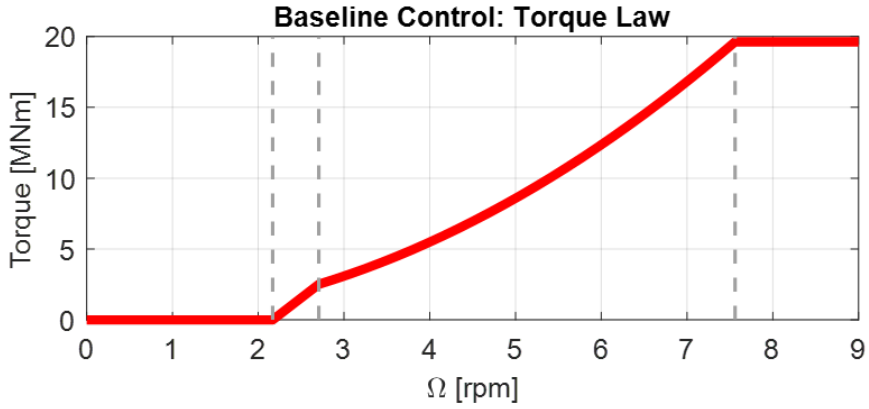

Figure 7: Example of Baseline generator torque controller

Blade Pitch Controller

Regarding the blade pitch controller, it regulates the generator speed in region 3 (where wind speed exceeds its rated value) to maintain it at its nominal value through a scheduled proportional-integral control (PI). In this region the torque is kept constant at its rated value:

$$T_{\text{gen}} = T_{\text{gen,r}} = \frac{P_r}{\Omega_r}.$$

Aerodynamic torque T_{aero} depends on wind speed, rotor speed and blade pitch, but assuming in this region rotor speed maintains its rated value Ω_r (this assumption can be made since the control objective is to track that value) and neglecting power to wind speed sensitivity, linearization around rated condition is:

$$T_{\text{aero}} \approx T_{\text{aero}}\left(U_{\text{wind},r}, \Omega_r, \theta_{bl,r}\right) + \left.\frac{dT_{\text{aero}}\left(U_{\text{wind}}, \Omega, \theta_{bl}\right)}{d\theta_{bl}}\right|_{\substack{U_{\text{wind}}=U_{\text{wind},r} \\ \Omega=\Omega_r}} \left(\theta_{bl} - \theta_{bl,r}\right) =$$

$$= \frac{P\left(U_{\text{wind},r}, \Omega_r, \theta_{bl,r}\right)}{\Omega_r} + \frac{1}{\Omega_r}\left.\frac{dP\left(U_{\text{wind}}, \Omega, \theta_{bl}\right)}{d\theta_{bl}}\right|_{\substack{U_{\text{wind}}=U_{\text{wind},r} \\ \Omega=\Omega_r}} \left(\theta_{bl} - \theta_{bl,r}\right) \tag{2}$$

where $U_{\text{wind},r}$ and $\theta_{bl,r}$ are rated wind speed and blade pitch. Once first is chosen, $\theta_{bl,r}$ is the value which leads to a steady state condition with extracted power equal to the rated one. So, aerodynamic torque expression becomes:

$$T_{\text{aero}} \approx \frac{P_r}{\Omega_r} + \frac{1}{\Omega_r}\frac{dP}{d\theta_{bl}}\Delta\theta_{bl} \tag{3}$$

Where $\Delta\theta_{bl}$ represents a small perturbation of the blade pitch angle about its linearization point $\theta_{bl,r}$. By expressing the blade-pitch regulation starting from the speed perturbation with a proportional-integrative control law (PI), it is possible to write:

$$\Delta\theta_{bl} = K_P \Delta\Omega + K_I \int_0^t \Delta\Omega \, dt \tag{4}$$

Where K_P is the proportional gain and K_I the integrative gain; $\Delta\Omega$ represents a small perturbation of rotor speed about its rated value: $\Delta\Omega = (\Omega - \Omega_r)$. Combining the last equations with the equilibrium equation of the rotor around its rotation axis ($T_{\text{aero}} - T_{\text{gen}} = I_{\text{eq}}\dot{\Omega}$), it is possible to obtain, once defined $\Delta\Omega = \dot{\delta}$, the following relation:

$$\frac{P_r}{\Omega_r} + \frac{1}{\Omega_r}\frac{dP}{d\theta_{bl}}\left(K_P\dot{\delta} + K_I\delta\right) - \frac{P_r}{\Omega_r} = I_{eq}\ddot{\delta} \tag{5}$$

Which can be rearranged as:

$$I_{eq}\ddot{\delta} + \left[-\frac{dP}{d\theta_{bl}}\frac{K_P}{\Omega_r}\right]\dot{\delta} + \left[-\frac{dP}{d\theta_{bl}}\frac{K_I}{\Omega_r}\right]\delta = 0 \tag{6}$$

That in the canonical form becomes:

$$M\ddot{\delta} + C\dot{\delta} + K\delta = 0 \tag{7}$$

With:

$$M = I_{eq}, \quad C = \left[-\frac{dP}{d\theta_{bl}}\frac{K_P}{\Omega_r}\right], \quad K = \left[-\frac{dP}{d\theta_{bl}}\frac{K_I}{\Omega_r}\right]$$

Now it is possible to choose proportional and integral gains in order to obtain desired characteristics of the blade pitch control. Its characteristics directly depend on natural frequency and damping ratio:

$$\omega_n = \sqrt{\frac{M}{K}}, \quad \zeta = \frac{C}{2M\omega_n} \tag{8}$$

Once defined ω_n and ζ, expressions of proportional and integral gains become:

$$K_P = \frac{2I_{eq}\omega_n\zeta\Omega_r}{-\frac{dP}{d\theta_{bl}}} = \frac{K'_P}{\frac{dP}{d\theta_{bl}}}, \quad K_I = \frac{I_{eq}\omega_n^2\Omega_r}{-\frac{dP}{d\theta_{bl}}} = \frac{K'_I}{\frac{dP}{d\theta_{bl}}} \tag{9}$$

The term $\frac{dP}{d\theta_{bl}}$ is the power to pitch sensitivity, which depends on wind speed and blade pitch (related each other as previously mentioned) adopted during linearization. So, to always have the same system characteristic (ω_n and ζ), proportional and integral gains must vary with a variation of blade pitch and so of wind speed. Figure 8 below shows power to pitch sensitivity with respect

to blade pitch; as can be seen there, it can be well approximated with a quadratic regression, through which the quadratic form that minimizes sum of square error is computed. Thanks to this regression, power to pitch sensitivity expression becomes of the form:

$$\frac{dP}{d\theta_{bl}} \approx c_1\theta_{bl}^2 + c_2\theta_{bl} + c_3 \tag{10}$$

$\frac{dP}{d\theta_{bl}}$ is the power to pitch sensitivity and $c_1\,(W/\text{deg}^3)$, $c_2\,(W/\text{deg}^2)$ and $c_3\,(W/\text{deg})$ are the coefficients of its quadratic regression.

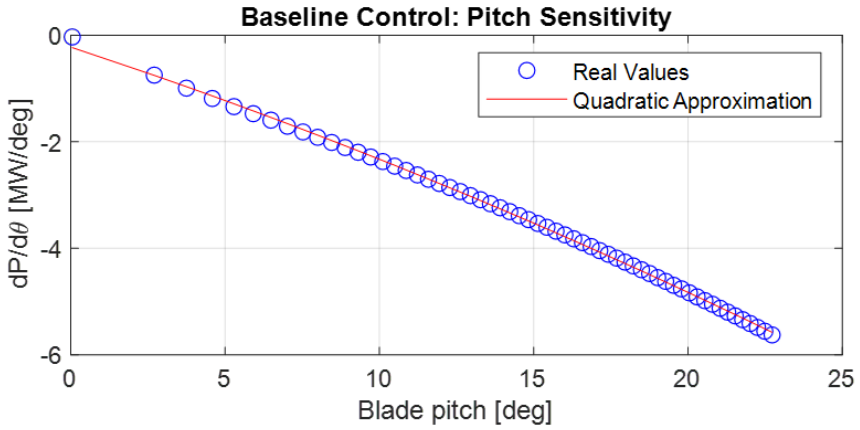

Figure 8: PowerToPitch Sensitivity

This approximation will make the calculation of controller gains computationally less demanding during simulation.

ROSCO

ROSCO controller (Reference Open-Source COntroller for fixed and floating offshore wind turbines) was developed by researchers at the Delft University of Technology [1] to provide a modular reference wind turbines controller that represent industry standards and performs comparably or better than existing reference controllers, such as baseline, discussed in previous section. The primary functions of the controller are still to maximize power in below-rated operations and to regulate rotor speed in above-rated ones, moreover, it also provides additional modules which can improve control performances. ROSCO controller, as well as Baseline and most of other conventional ones, consists of two methods of actuation: generator torque and collective blade pitch. Strategies of actuation are commonly separated into four main regions, with transition logic between them. Regions 1 and 4 correspond to below cut-in and above cut-out wind speed conditions, these regions are generally out of interest for standard control purposes (performances optimization) and so they will not be further discussed below. In region 1 generator torque is set to zero to allow the wind to accelerate the rotor for start-up. In this region, no power is extracted. In region 4 blades are pitched to reduce thrust force to zero (feathering position).

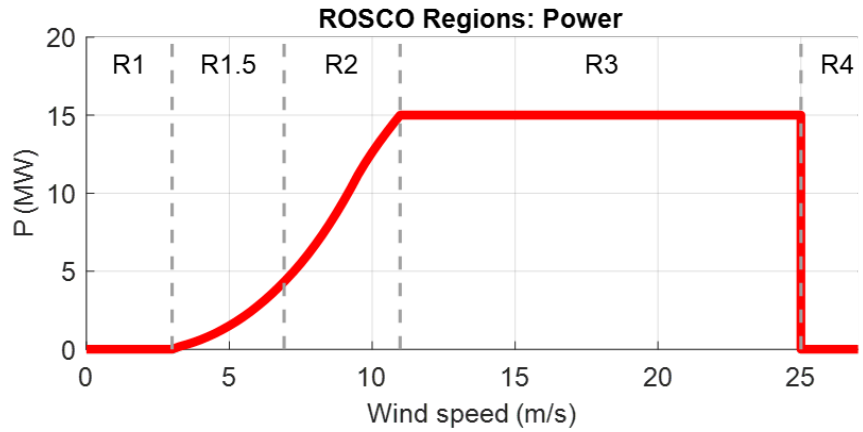

Figure 9: ROSCO Power Curve

Control strategies for regions 1.5, 2 and 3 are highly like those ones adopted in Baseline control. Region 2 is when wind speed is below rated condition, here main goal is power extraction maximization. To do so, two methods can be used, a quadratic law (as in Baseline controller) of generator torque with respect to rotor angular speed or a tip speed ratio (TSR) tracking to maintain the latter at its optimal value (in this case a wind speed estimation is needed). Region 3 is when wind speed is above rated condition, here blade pitch is regulated to maintain rotor speed at its rated value and to stabilize platform (for offshore floating wind turbines, through floating feedback module), while generator torque is kept constant at its rated value. Region 1.5 is a transition region from cut-in wind speed and region 2. Here generator torque is regulated to maintain a defined minimum rotor speed and blades are pitched to compensate resulting high values of TSR to improve power extraction.

ROSCO Implementation

Controller implementation starts from aerodynamic torque expression T_{aero} and rotor equilibrium:

$$T_{\text{aero}} = \frac{1}{2}\rho A_D C_P(\lambda, \theta_{bl})\frac{U_\infty^3}{\Omega} \tag{11}$$

$$\dot{\Omega} = \frac{T_{\text{aero}} - T_{\text{gen}}}{I_{\text{eq}}} \tag{12}$$

where I_{eq} is rotor inertia, ρ air density, A_D rotor area, C_P power coefficient, and U_∞ undisturbed wind speed. Linearizing (11) at a steady-state operational point:

$$\Delta T_{\text{aero}} = \Gamma_\Omega|_{\text{op}}\Delta\Omega + \Gamma_{\theta_{bl}}|_{\text{op}}\Delta\theta_{bl} + \Gamma_U|_{\text{op}}\Delta U \tag{13}$$

where $\Gamma_\Omega|_{\text{op}} = \partial T_{\text{aero}}/\partial\Omega|_{\text{op}}$, $\Gamma_{\theta_{bl}}|_{\text{op}} = \partial T_{\text{aero}}/\partial\theta_{bl}|_{\text{op}}$, and $\Gamma_U|_{\text{op}} = \partial T_{\text{aero}}/\partial U|_{\text{op}}$. "op" denotes the steady-state operational point at which linearization is made. Equation 12 can then be

rewritten as (Δ denotes the perturbation from steady state value "op" and $X_{\text{op}} = \lambda_{\text{op}}, \theta_{\text{bl,op}}$):

$$\Delta\dot{\Omega} = A(\boldsymbol{X}_{\text{op}})\Delta\Omega + B_{T_{\text{gen}}}\Delta T_{\text{gen}} + B_{\theta_{bl}}(\boldsymbol{X}_{\text{op}})\Delta\theta_{bl} + B_U(\boldsymbol{X}_{\text{op}})\Delta U \quad (14)$$

With:

$$A\left(\boldsymbol{X}_{\text{op}}\right) = \frac{1}{I_{\text{eq}}}\frac{\partial T_{\text{aero}}}{\partial\lambda}\frac{\partial\lambda}{\partial\Omega}$$

$$\frac{\partial T_{\text{aero}}}{\partial\lambda} = \frac{1}{2}\rho A_{\text{D}} R U_{\text{op}}^2 \frac{1}{\lambda_{\text{op}}^2}\left(\frac{\partial C_{\text{p}}}{\partial\lambda}\lambda_{\text{op}} - C_{\text{p,op}}\right)$$

$$\frac{\partial\lambda}{\partial\Omega} = \frac{R}{U_{\text{op}}}, \quad \left(\lambda = \frac{\Omega R}{U}\right)$$

$$B_{T_{\text{gen}}} = -\frac{1}{I_{\text{eq}}}$$

$$B_{\theta_{bl}}\left(\boldsymbol{X}_{\text{op}}\right) = \frac{1}{2I_{\text{eq}}}\rho A_{\text{D}} R U_{\text{op}}^2 \frac{1}{\lambda_{\text{op}}^2}\left(\frac{\partial C_{\text{p}}}{\partial\theta_{bl}}\lambda_{\text{op}}\right)$$

All derivatives are calculated at "op" conditions; ΔU, difference between actual wind speed and wind speed at linearization point, is considered equal to zero during control tuning, that is computation of control gains. Both generator torque and blade pitch controllers are PI controllers, generically defined as:

$$y = K_P\, u + K_I \int_0^T u\, dt \quad (15)$$

Where u represents the input and y the output, while K_P and K_I are respectively the proportional and integral gains. Generator torque controller has as input and output:

$$u = -\delta\Omega, \quad y = \Delta C_{\text{gen}} \quad (16)$$

Blade pitch controller has as input and output:

$$u = -\delta\Omega, \quad y = \Delta\theta_{bl} \quad (17)$$

$\delta\Omega$ is defined as a perturbation from the reference speed:

$$\Omega(t) = \Omega_{\text{ref}} + \delta\Omega \longrightarrow -\delta\Omega = \Omega_{\text{ref}} - \Omega(t) \quad (18)$$

While ΔC_{gen} and $\Delta\theta_{bl}$ are perturbations from steady state values:

$$\theta_{bl}(t) = \theta_{bl_{\text{op}}} + \Delta\theta_{bl}, \quad C_{\text{gen}}(t) = C_{\text{gen}_{op}} + \Delta C_{\text{gen}} \quad (19)$$

Now, defining $\Delta\Omega_{\text{ref}} = \Omega_{\text{ref}} - \Omega_{\text{op}}$ (assumed =0, since "op" point is chosen at a steady state condition with $\Omega_{\text{op}} = \Omega_{\text{ref}}$), we can combine equation 14 with above definitions to obtain a differential equation that relates $\Delta\Omega = \Omega - \Omega_{\text{op}}$ and $\Delta\Omega_{\text{ref}}$. Then, if the Laplace transform of

this equation is considered, we arrive to two closed-loop transfer functions (one for the generator torque module and the other for the blade pitch one) in the form:

$$H(s) = \frac{\Delta\Omega(s)}{\Delta\Omega_{\mathrm{ref}}(s)} = \frac{B\left(K_P\left(x_{\mathrm{op}}\right)s + K_I\left(x_{\mathrm{op}}\right)\right)}{s^2 + \left(BK_P\left(x_{\mathrm{op}}\right) - A\left(x_{\mathrm{op}}\right)\right)s + BK_I\left(x_{\mathrm{op}}\right)} \quad (20)$$

Where B is $B_{T_{\mathsf{gen}}}$ or $B_{\theta_{bl}}$, depending on which module is considered, since when generator torque loop is considered, $\Delta\theta_{bl}$ is set to zero and, when blade pitch loop is considered, ΔT_{gen} can be equal to zero or $B_{T_{\mathsf{gen}}}$ can be englobed in A. Moreover, in both cases we consider $\Delta U = 0$. $H(s)$ is a simple second order system whose characteristics are strictly related to natural frequency and damping ratio of its canonical form. They can be defined, in order to reach desired performance, choosing values of proportional and integral gains. If we call ω_n the natural frequency and ζ the damping ratio, K_P and K_I expressions (varying with operational steady state point) are:

$$K_P = \frac{1}{B\left(\boldsymbol{x}_{\mathrm{op}}\right)}\left(2\zeta\omega_n + A\left(\boldsymbol{X}_{\mathrm{op}}\right)\right) \quad (21)$$

$$K_I = \frac{\omega_{\mathrm{n}}^2}{B\left(\boldsymbol{X}_{\mathrm{op}}\right)} \quad (22)$$

Once transfer function of generator torque and blade pitch closed loop has been defined, and once way through which PI controllers' gains are computed has been explored, we can focus, specifically, on the two different modules to investigate the reference speed signals adopted and how the scheduling of gains is performed, varying according to the conditions in which the system is.

Generator Torque Controller

Four different generator torque controllers are available in ROSCO, they are the possible combination between two methods for below wind speed operations and two methods for above wind speed conditions. Regarding below rated operations, to maximize extracted power at each wind condition, a quadratic low of generator torque with respect to rotor angular speed can be adopted. In this section we omit exploitation of this method since is the same adopted in Baseline controller. Alternatively, a tip speed ratio tracking to maintain it at its optimal value can be adopted. If the wind speed can be measured or estimated accurately, a generator torque controller can be designed to maintain the λ_{opt} and maximize power capture, so reference rotor angular speed becomes:

$$\Omega_{\mathsf{ref},\tau} = \frac{\lambda_{\mathsf{opt}}\hat{U}}{R} \quad (23)$$

Where subscript τ indicates the reference speed of torque controller and $\hat{U}$ is the estimated wind speed. From equations 14, 21 and 22, it can be seen that integral gain K_I of generator torque controller is constant, whereas A, so proportional gain K_P, are both dependent on U (wind speed). However, it was found that fixing $K_P = K_P(U = U_{\mathsf{rated}})$ does not negatively affect

power production. Regarding the two existing methods for above rated conditions, first of them considers a constant generator torque, defined as:

$$T_{\text{gen,ar}}(t) = T_{\text{rated}} = \frac{P_{\text{rated}}}{\Omega_{\text{rated}}} \quad (24)$$

Where subscript "ar" means "above rated". On the other hand, the second strategy considers a constant extracted power equal to its rated value, so generator torque is defined as:

$$T_{\text{gen,ar}}(t) = \frac{P_{\text{rated}}}{\Omega} \quad (25)$$

Blade Pitch Controller

Main goal of blade pitch controller is keeping rotor angular speed at its rated value, so reference speed is (both in below rated and above rated conditions):

$$\Omega_{\text{ref},\theta_{bl}} = \Omega_{\text{rated}} \quad (26)$$

Where subscript θ_{bl} means we refer to blade pitch controller. In below rated conditions, generator speed is lower than rated value, so $-\delta\Omega = \Omega_{\text{ref}} - \Omega > 0$ and, since gains are normally negative, θ_{bl} is saturated at its minimum value, defined by an additional module of ROSCO controller which will be discussed later. According to equations 21 and 22, to find controllers gain values, $B_{\theta_{bl}}(X_{\text{op}})$ and $A(X_{\text{op}})$ should be computed. They change for any operation point at which system is linearized, so they are function of $X_{\text{op}} = \{\lambda_{\text{op}}, \theta_{bl\,\text{op}}\}$. Linearization point can be the optimal steady state values chosen during strategy definition, for which there is a unique relationship between λ_{op} and $\theta_{bl\,\text{op}}$. For this reason, $B_{\theta_{bl}}$ and A can be expressed with respect to $\theta_{bl\,\text{op}}$, so gains' values can be scheduled with θ_{bl} as parameter.

Additional Control Modules

In this section principal additional modules are briefly discussed to understand their functions and how they modify control output; for more information it is possible to consult [1]. They are:

- **Wind speed estimator**: This module estimates wind speed used for TSR tracking in the generator torque controller. The employed algorithm is based on a continuous-discrete Kalman filter, which exploits the system model, a wind auto-regressive model, and other information like covariance matrices based on the expected wind field and the measurement confidence of rotor speed to estimate a mean wind speed across the rotor area at each time.
- **Set Point Smoothing**: Generator torque and blade pitch controllers will normally conflict with each other in near-rated operation due to incompatible reference rotor speed. To avoid this, a set point smoother can be employed; it shifts the speed reference signal of the inactive controller while the active one works. As an example, at above rated conditions,

the torque controller is the inactive one and vice versa. If TSR tracking were to be adopted for the torque generator, then the reference speed at high wind speeds would be higher than the one actually wanted (rated one), so the smoother brings the reference towards the rated speed, and the resulting torque approaches the rated one, which is intended by adopting a constant torque strategy under these conditions.

- **Minimum pitch Saturation**: This module defines a minimum value of blade pitch angle which will be used as a saturation limit during control operations. It mainly modifies expected blade pitch values in region 1.5 and near-rated conditions and leads to two effects:
  - **Peak shaving**: Near-rated conditions thrust values reach their highest, since below-rated wind speed is lower and above-rated conditions involve blade pitching to reduce that force. To limit loads, the minimum pitch module imposes non-zero pitch angles even below rated wind speed, near that value.
  - **Power maximization in low wind**: In region 1.5, as mentioned in the control region section, a minimum value of rotor speed is imposed, so at low wind speeds, TSR deviates far from its optimal value. To compensate for this fact and to increase the power coefficient value in this condition, blade pitch is set to be greater.

- **Floating offshore wind turbine feedback**: This module is designed for FOWTs (Floating Offshore Wind Turbines) and introduces a new term in the PI blade pitch controller, which becomes:

$$\Delta\theta_{bl} = -k_{\mathrm{P}}\delta\Omega - k_{\mathrm{I}}\int_0^T \delta\Omega\,\mathrm{d}t + k_{\theta_{\mathrm{bl,float}}}\dot{x}_t \quad (27)$$

  The additional term is tower-top velocity $\dot{x}_t$ multiplied by $k_{\theta_{\mathrm{bl,float}}}$ gain. The latter is chosen from a manipulation of the rotor equilibrium equation and structure pitch equation, in which expressions of thrust and power coefficients appear. The aim is to find gain values that reduce the rotor angular acceleration's sensitivity to tower speed, mitigating the structure pitch effect on rotor aerodynamic torque. This expedient increases the average extracted power and stabilizes the structure.

In this case, TSR tracking was chosen for torque control at wind speeds lower than the nominal one, with a constant torque equal to nominal in above-rated conditions. Furthermore, wind speed is assumed to be a priori known, so the Kalman filter in the estimation module will not be exploited.

3.2.3 Aerodynamic loads

The aerodynamic forces and torques produced by the wind-blade interaction can be calculated during the dynamic simulation using two different methods. The first consists of solving the BEM (Blade Element Momentum) problem for each blade node to which an airfoil type is associated and taking into account the velocity of the system. The resolution is based on [9] and [8]. At each instant and for each node there is an iterative procedure that calculates the linear and

tangential induction factors. The second method (more computationally efficient but also less accurate) involves the use of look-up tables whose outputs are the forces and torques at the base of the blades. The inputs are the actual velocity of the wind at certain points of the blades, the rotor speed and the pitch angle of the blades. The rotor speeds and blade pitch angle for which loads are calculated during pre-processing vary as the wind speed varies; in particular, for each wind speed, a range of these two quantities is considered around their stationary value for that wind. These stationary values are calculated in the scripts 'Steady_States', while look-up tables are created in the 'AeroLoads' scripts. The aerodynamic forces do not take into account the flexibility of the blade (rigid body assumption).

3.3 Mooring

Mooring systems can be simulated in different ways, some of them introduced with MOST. The first two are based on the use of stiffness and damping matrices or the use of an external open-source software (MoorDyn) that solves a lumped-mass model. MOST introduces the possibility of simulating moorings with a quasi-static non-linear model based on the catenary equations, analogous to the open-source code MAP++. In addition, it is also possible to use look-up tables obtained during preprocessing that solve the forces and torques for a number of combinations of fearlead positions, again using the quasi-static model mentioned above. In this case, the forces and torques due to the moorings are determined via 6 different look-up tables that have as input the 6 degrees of freedom of the moored body. The outputs (Fx, Fy, Fz, Mx, My and Mz, i.e. the mooring loads) are contained in a data structure called 'moor_LUT' and created via the script 'MooringLUTMaker.m'.

4 User Manual

This section provides the explanations for using MOST, starting with the creation of all the data needed for the dynamic simulation (Pre-processing) and then moving on to how to define the inputs needed for it (Main input file). We will then proceed with an overview of the Simulink model (Simulink model) and finally explain how the post-processing phase works (Post-processing). This guide is mainly focused on the features introduced in the wecSim environment by MOST, so please refer to the dedicated guide for the features already present there.

4.1 Pre-processing

In the pre-processing phase, it is possible to create the data which, depending on the settings chosen for the dynamic simulation, may be required for modeling the wind field, wind turbine and mooring system and for creating the hydrodynamic data required for the simulation of floating bodies. The following paragraphs will explain which scripts to use and how to define their settings, describing the inputs that can be changed.

4.1.1 Wind

As mentioned in the theory it is possible to have two types of wind field, the first constant in space for every point in the domain, the second is a turbulent wind field varying in time and space. In this second case, it is necessary during pre-processing to create the aforementioned wind field. To do this, the open-source software TurbSim [5] created by NREL (National Renewable Energy Laboratory) is used. Using this software, a turbulent wind field can be created (the degree of turbulence can be chosen, for example), the output is the wind speed (vector with components along x-y-z) corresponding to each point on a three-dimensional grid and at each instant. Among the TurbSim settings, it is possible to define the average angles of vertical inclination (uptilt or elevation) and horizontal inclination (skew or azimuth), which remain constant throughout. In order to expand the potential by being able to vary these angles over time, they are kept null during the creation of the wind field and are instead modified a posteriori in the 'RunTurbsim.m' script. In particular, it is possible to define these angles for several instants of time (time breakpoints), then the entire wind field will be rotated to meet this setting and interpolating linearly at intermediate instants of time, so that a change in the mean wind direction can be simulated during a single dynamic simulation. Operationally, it is necessary to go to the folder 'mostData/turbSim' and run the script 'RunTurbsim.m'. In order to obtain the wind file with the desired characteristics, it is necessary to act in the 'SETTINGS' section and/or modify the input file read by the software in the folder and called 'TurbsimInputFile.txt'. The variables that can be modified in the script and their explanation are given below:

Name	Description
`WINDvector`	Mean wind speeds vector (for each value a wind field file will be created)
`Xgrid`	Wind field domain (surge-component) extension
`time_breakpoints`	Time breakpoints for with wind characteristics are defined
`elevations`	Wind speed elevation angles at time breakpoints
`azimuths`	Wind speed azimuth angles at time breakpoints
`filename_InputTurbsim`	Name of the main TurbSim input file
`deleteOut`	Flag to choose if temporary Turbsim output files must be deleted

Table 1: RunTurbsim.m inputs

By modifying these variables, it is only possible to act, as far as inputs to the TurbSim software are concerned, on the average (long surge) wind speed. That is, the default file 'TurbsimInputFile.txt' will be used in which only this input is modified. If, however, you wish to change other settings, you must edit this file directly. Below is an explanation of the main settings that can be changed (for further information, please refer to the TurbSim user guide in the folder):

Name	Description
`NumGrid_Z`	Vertical grid-point matrix dimension
`NumGrid_Y`	Horizontal grid-point matrix dimension

TimeStep	Time step [s]
AnalysisTime	Length of analysis time series [s]
UsableTime	Usable length of output time series [s]
HubHt	Hub height [m]
GridHeight	Grid height [m]
GridWidth	Grid width [m]
VFlowAng	Vertical mean flow (uptilt) angle [deg]
HFlowAng	Horizontal mean flow (skew) angle [deg]
TurbModel	Turbulence model ("IECKAI"=Kaimal, "IECVKM"=von Karman, "GP_LLJ", "NWTCUP", "SMOOTH", "WF_UPW", "WF_07D", "WF_14D", "TIDAL")
IECstandard	Number of IEC 61400-x standard (x=1,2, or 3 with optional 61400-1 edition number)
IECturbc	IEC turbulence characteristic ("A", "B", "C" or the turbulence intensity in percent)
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)
ETMc	IEC Extreme Turbulence Model "c" parameter [m/s]
WindProfileType	Wind profile type ("JET";"LOG"=logarithmic;"PL"=power law;"H2L"=Log law for TIDAL spectral model;"IEC"=PL on rotor disk, LOG elsewhere)
RefHt	Height of the reference wind speed [m]
PLExp	Power law exponent [-]
Z0	Surface roughness length [m]
PC_UW	Hub mean u'w' Reynolds stress
PC_UV	Hub mean u'v' Reynolds stress
PC_VW	Hub mean v'w' Reynolds stress

Table 2: TurbsimInputFile.txt inputs

4.1.2 Wind turbine

In this section, the way in which all the wind turbine data required for the dynamic simulation is created is explained. The data required are the geometric characteristics of the turbine to be simulated, as well as the inertial properties and, in the event that one wishes to visualise the components during simulation, the CAD files of the various components (tower, nacelle, hub and blades). In addition, it is necessary to have the geometric data of the blades (values varying the distance from the chord hub, twist angle and values related to pre-bending) and the characteristics of the airfoils comprising them, i.e., for each type of airfoil, the drag, lift and torque coefficients as the angle of attack varies. So far, all this information is already available for three different reference turbines, with power ratings of 5, 10 and 15 MW. The sources from

which they were obtained are: 5MW, 10MW and 15MW. Creating the data for the simulation of these three reference turbines can easily be done by navigating to the 'mostData/windTurbine' folder and launching the 'WindTurbineMaker.m' script, in which it is possible to define for which turbine the data should be built. By launching this script, other scripts are executed, each one dedicated to the creation of files with a .mat extension, which will then be used for the dynamic simulation. The name and order of execution of the scripts is as follows:

1. WTproperties_(WindTurbine_type).m
2. BladeData_(WindTurbine_type).m
3. Steady_States_(WindTurbine_type).m
4. Controller_(WindTurbine_type).m
5. AeroLoads_(WindTurbine_type).m

where "WindTurbine_type" is the name of the variable whose content corresponds to the chosen turbine.

In each of these scripts, it is possible to change certain settings. In the following paragraphs, we will see in detail which scripts are called up, how they work and which settings can be changed. If, on the other hand, a turbine other than the three mentioned above is to be used in the dynamic simulation, it is necessary to find the necessary data and place it in the appropriate folders in the same way as was done for the three already present.

Properties

The wind turbine is modeled as a multi-body system including the tower, the nacelle, the hub, and the blades. The properties of each wind turbine component are defined in two structures that can be generated using the provided "WTproperties" and "BladeData" MATLAB codes. the geometric and inertial properties of the turbine components are defined. For each of them the mass and inertia are defined, in addition, the following other variables must be entered (see Figure 10 for a better comprehension):

Name	Description
tower.offset	Tower offset position relative to sea water level (m)
tower.height	Tower height (m)
tower.cog_rel	Tower relative center of mass (relative to tower offset) (m)
nacelle.cog_rel	Nacelle relative center of mass (relative to tower top) (m)
nacelle.Twr2Shft	Twr2Shft (deg)
nacelle.tiltangle	Tilt angle (deg)
hub.overhang	Overhang (m)
hub.height	Hub height (m)
hub.Rhub	Hub radius (m)

hub.precone	Precone angle (deg)
blade.cog_rel	Blades relative center of mass (relative to hub center) (m)
blade.bladeDiscr	Blade discretization nodes to average the wind speed
gen_eff	Generator efficiency
geometryFile	CAD file path with respect to wecSimImputFile.m path

Table 3: WTproperties scripts inputs

Once these dimensions are known, the positions of the centre of mass of each component in the inertial reference frame are calculated (origin at sea level and at the centre of the tower, as far as the horizontal plane is concerned), as well as the total mass and inertia.

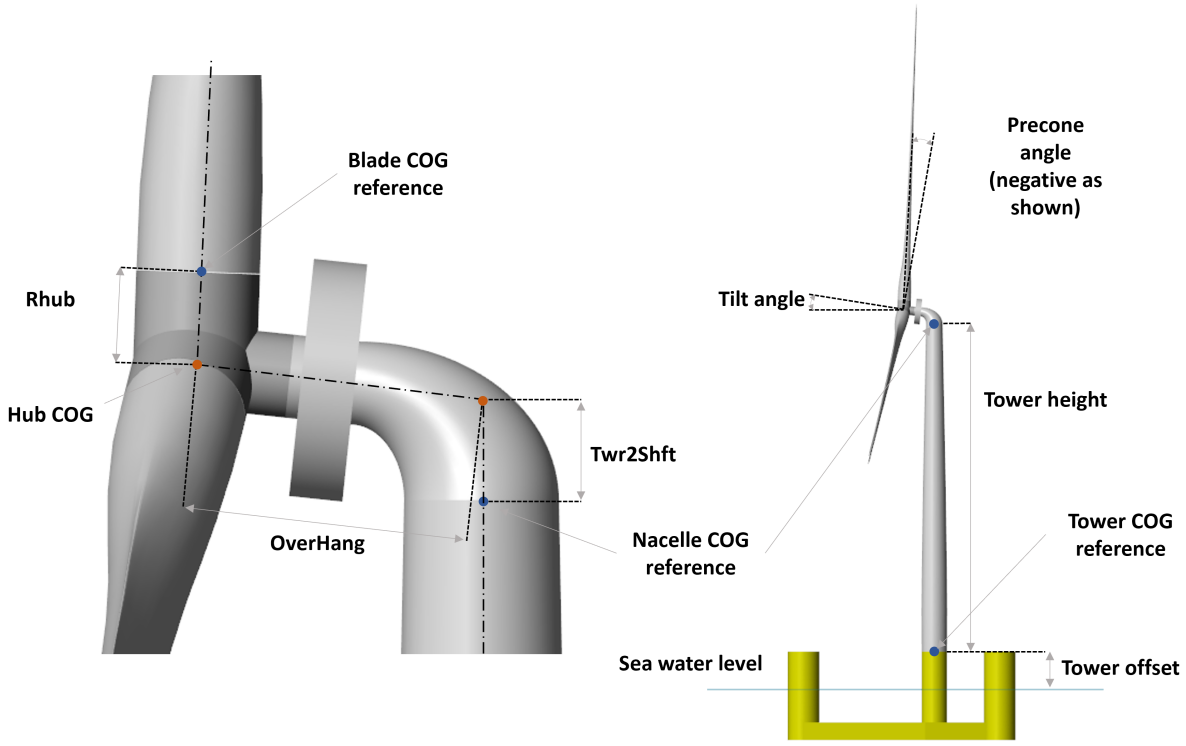

Figure 10: Wind Turbine Geometry

In the first, the variables concerning the blades are defined, specifically (see Figure 11 for a better comprehension):

Name	Description
radius	Blade radius values for which other properties are defined
BlCrvAC	Value, for each specified radius, of the pre-bending distance out of the rotor plane

BlCrvAng	Value, for each specified radius, of the pre-bending angle out of the rotor plane
BlSwpAC	Value, for each specified radius, of the pre-bending distance in the rotor plane
twist	Value, for each specified radius, of the twist angle
chord	Value, for each specified radius, of the chord
airfoil_index	Index of the airfoil type corresponding to each specified radius
airfoil	Matrix containing, for each type of airfoil existing, the values of lift, drag, and torque coefficients as a function of angle of attack

Table 4: BladeData scripts inputs

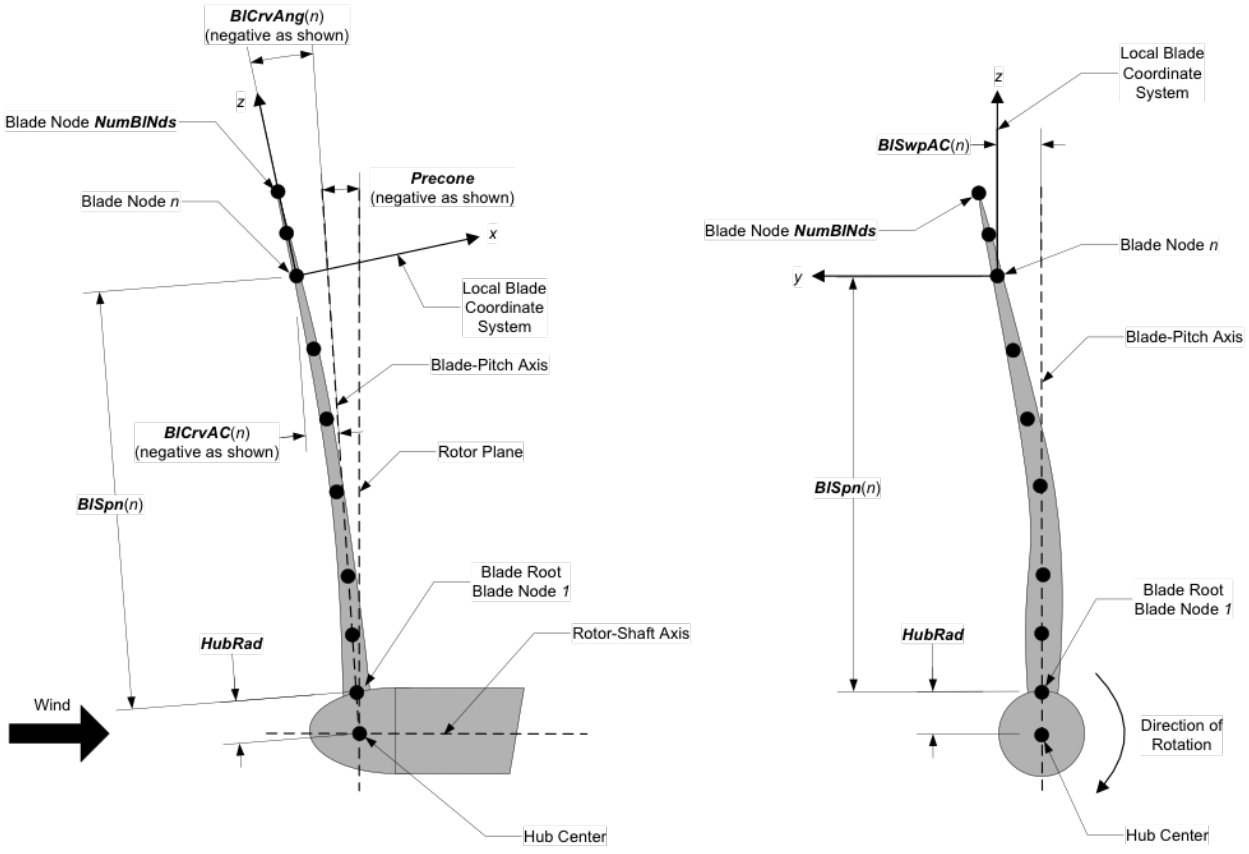

Figure 11: Blade geometry

Control

In MOST there is the possibility of using two different wind turbine control logic (Baseline [4] and ROSCO [1]) and as explained in the theory the steps to be taken in order to obtain the data needed for their simulation are the calculation of the stationary values and the calculation of the controller gains. The first task is performed by the scripts "Steady_States" in the sub-folder "mostData/windTurbine/control". Specifically, through this, the stationary values of power, rotor speed, thrust force, generator torque and collective blade pitch angle are computed for both of the aforementioned control logic. The following are the main variables to be specified in the script:

Name	Description
Prated	Value of power produced under nominal conditions and under conditions where the wind speed is greater than the nominal one
v_cutin	Wind speed at which power begins to be produced (and therefore at which the generator torque becomes non-zero)
v_cutout	Wind speed above which no power is produced, and the blades rotate to a safe position (feather condition)
v_rated_try	Rated first try wind speed, meaning that the actual wind speed (probably close to this) will be calculated later
omega_rated_try	Rated first try rotor speed, meaning that the actual one (probably close to this) will be calculated later
v_discr	Wind speed discretization, which indicates how many stationary values will be calculated
omega_min	Minimum allowed value of the rotor speed (setting used only for the calculation of stationary values related to the ROSCO controller)
vw_R2in_try	Boundary wind speed value between zone 1.5 (zone with constant and equal to minimum rotor speed) and zone 2 (zone with optimal tip speed ratio), this value is used only for the ROSCO controller
max_Thrust_factor	Ratio of the maximum allowed thrust to what there would be if no limits were imposed

Table 5: SteadyStates scripts inputs

The script calculates different stationary values according to the control logic because of their diversity. Specifically, only the ROSCO controller imposes an upper limit for the thrust force, so when the wind speed is close to the nominal one (where the force peak occurs), the blade pitch value will be slightly higher to reduce the thrust and comply with the imposed limits. The second difference is that in the Baseline controller, no minimum rotor speed is imposed, which is also the case of some turbine types for what concerns the ROSCO controller. The first step performed in the code is the calculation of the actual nominal conditions (rated wind speed, rotor speed and

blade pitch angle): by means of a constrained optimization, the values of wind speed, rotor speed and blade pitch angle are sought such that the first two are as close as possible to those set for the first attempt, with the constraint of having a thrust not exceeding the maximum and a power output equal to the rated one. In the case of the Baseline controller, the first constraint does not apply, in the case of the ROSCO controller, on the other hand, we first calculate the nominal conditions without the thrust constraint, then calculate the maximum thrust by multiplying the nominal one by the thrust factor defined in the settings and then repeat the calculation to find the correct nominal values. The optimization relies on a function that calculates the aerodynamic forces at the hub by solving the BEM (Blade Element Momentum) theory, for more information on how this function works see ([9] , [8]).

The second step, performed only in the case of ROSCO, involves finding the wind speed for which transition from zone 1.5 to zone 2 (see [1]) occurs. In both zones it is desired to maximize power, but in zone 1.5 is where there is the additional constraint of minimum rotor speed. Here, to maximize power, the rotor speed would need to be less than the minimum rotor speed, and to partially compensate for the resulting power deficit, a blade pitch angle greater than zero is used. The search for the frontier wind speed is done by an optimization that looks for the wind speed for which the difference between the rotor speed that maximizes power without imposing constraints equals the minimum wind speed. To find the rotor speed that maximizes power for a given wind speed, a second nested optimization is used.

Finally, the last step involves calculating the stationary values as the wind speed changes. It is performed by a constrained optimization through which the rotor speed and blade pitch values are sought such that the power produced is maximized while maintaining it at or below the rated power and respecting the maximum thrust limit. Once the rotor speed and blade pitch values have been found for each wind speed analyzed, the steady-state values of the other quantities of interest (power, thrust, and generator torque) are evaluated.

Once the steady-state values for the two control logic have been calculated, it is possible to build the data structures needed for controller simulation by running the "Controller.m" script in the "mostData/windTurbine/control" sub-folder. In this script a few settings have to be defined, which can refer to both logic or just the Baseline or ROSCO controller.
The common settings are as follows:

Name	Description
torqueMaxRate	Maximum allowable torque rate
thetaMaxRate	Maximum allowable blade pitch rate
omegaFilter	Values needed to define the state space used to filter the rotor speed before providing this as input to the controllers

Table 6: Baseline-ROSCO common inputs

The settings only valid for ROSCO are:

Name	Description

wn_C	Natural frequency of the electric generator torque control loop
csi_C	Damping ratio of the electric generator torque control loop
wn_theta_ROSCO	Natural frequency of the blade pitch control loop
csi_theta_ROSCO	Damping ratio of the blade pitch control loop
kb and kt	Constants used in the "Set Point Smoothing" module
KV	Gain related to the velocity (along the surge) of the nacelle, used to control floating wind turbines
windFilter	Values needed to define the state space used to filter the wind speed before providing this as input to the controller
pitchFilter	Values needed to define the transfer function used to filter the nacelle speed before providing this as input to the controller
SPSFilter	Values needed to define the transfer function used as a filter in the "Set Point Smoothing" module

Table 7: ROSCO inputs

The settings only valid for Baseline are:

Name	**Description**
wn_theta_BL	Natural frequency of the blade pitch control loop
csi_theta_BL	Damping ratio of the blade pitch control loop

Table 8: Baseline inputs

Regarding the Baseline controller, at the operational level, the torque law is simply computed by creating a bi-univocal relationship between the steady-state (as wind speed changes) values of rotor speed and generator torque. As for the blade pitch loop, at first the value of K'_P and K'_I are calculated (see [4]), after which the power to pitch sensitivity, as a function of blade pitch angle, is computed for each stationary point. To do this, centered finite differences are used by calculating power via a function that solves the aerodynamics via BEM theory. Finally, we perform quadratic regression of $\frac{dP}{d\theta_{bl}}$ so that we have, in simulation, a simple relationship between blade pitch and power-to-pitch sensitivity.

As for the ROSCO controller, however, at the operational level, in the script the A and $B_{\theta_{bl}}$ matrices are calculated for each wind speed for which stationary values were computed through centered finite differences; regarding the $B_{T_{gen}}$ matrix, it is calculated only once since it is constant (see [1]). Once the matrices are known, the values of K_P and K_I for the two controls (generator torque and blade pitch) are calculated. Finally, the minimum allowable blade pitch value is calculated using an optimization procedure; specifically, for each wind speed in region 3 (a rotor speed equal to the nominal one is assumed in this region), the blade pitch angle such that the maximum thrust occurs is found. It represents the minimum angle value that can be imposed, below which there will be a thrust greater than the maximum allowed.

Regarding yaw control, as mentioned in the theory, what is implemented so far in MOST is a simple PID controller that adjusts the relative rotation speed between tower and RNA (Rotor Nacelle Assembly) based on the difference between average wind direction and angular position of the rotor. However, it is possible to easily act on the Simulink model in order to implement more advanced control logic. In the 'Controller' scripts, it is possible to define the characteristics of the PID controller mentioned above by acting on the variables described below:

Name	Description
maxYawRate	Maximum relative angular velocity between tower and nacelle
Kp	Proportional gain
Ki	Integral gain
Kd	Derivative gain
tau_system	Time constant of a first-order transfer function that takes into account, in a simplified manner, the delay in the system's rotation response due to its inertia

Table 9: Yaw controller inputs

Aerodynamic loads

As mentioned in the theory, there are two ways to calculate the aerodynamic forces acting on wind turbine blades, the first consists of solving a BEM (Blade Element Momentum) problem at each instant and for each blade section as specified in and in. The second involves solving this problem in pre-processing for a limited combination of configurations and creating look-up tables, which will be used during the numerical simulation. For this second case, therefore, it is necessary to create these look-up tables in the pre-processing phase; to do this, the 'AeroLoads' scripts in the 'mostData/windTurbine/aeroloads' folder must be run. In addition to entering the correct names of the previously created files to be used for calculation, the main variables to be acted upon and their explanation is given in the table below.

Name	Description
o_discr	Rotor speed discretization value
theta_discr	Blade pitch discretization value
o_A	Discretization range of rotor speed values around steady-state one (rad/s)
theta_A	Discretization range of blade pitch values around steady-state one (rad)

Table 10: "Aeroloads" inputs

The discretization range is used to determine the aerodynamic loads near the steady states, which include all cases that are likely to be reached during operating conditions.

4.1.3 Mooring

In the Simulink model, forces and torques due to moorings are determined through 6 different look-up tables having the 6 degrees of freedom surge, sway, heave, roll, pitch, and yaw as inputs. The breakpoints (related to the inputs) and the outputs (Fx, Fy, Fz, Mx, My and Mz, i.e., the mooring loads) are contained within a data structure called "moor_LUT" and created through the "MooringLUTMaker.m" script, in which the following variables are specified:

Name	Description
gravity	Gravity acceleration (m/s^2)
rho_water	Water density (kg/m^3)
depth	Water depth (m)
d	Mooring line diameter (m)
linear_mass	Linear mass (kg/m)
number_lines	Number of lines
nodes	Fairlead and anchor positions of the first line (m)
L	Mooring lines unstretched length (m)
EA	Sectional stiffness (N)
CB	Seabed friction coefficient

Table 11: "MooringLUTMaker.m" inputs

In addition, the user can specify the domain of the look-up tables, specifically:

Name	Description
X	Surge positions at which mooring loads are computed (m)
Y	Sway positions at which mooring loads are computed (m)
Z	Heave positions at which mooring loads are computed (m)
RX	Roll rotations at which mooring loads are computed (rad)
RY	Pitch rotations at which mooring loads are computed (rad)
RZ	Yaw rotations at which mooring loads are computed (rad)

Table 12: "MooringLUTMaker.m" inputs: look-up table domain

The code for generating the "moor_LUT" structure at first calculates the positions of the fairleads and anchors of the other lines, in accordance with the specified number and in an angularly equi-spaced manner, after which, for each combination of the inputs (surge, sway, heave, roll, pitch, and yaw) it calculates the new positions of the fairleads. Given these positions, for each line it performs a numerical optimization by which the vertical and the horizontal forces (along the projection of the line in the xy plane) are calculated by means of the typical catenary equations. Once the forces are obtained for each line, the resulting loads in the global reference frame are applied to the origin of the reference frame attached to the structure. In addition, the stiffness matrix linearized around a given position is computed, as well as the loads for that position. These may be useful if it is desired to simulate the mooring system using a simple stiffness matrix (see WEC-Sim online documentation for more details).

4.1.4 Hydrodynamic

To simulate floating bodies in WEC-Sim, a file in .h5 format with all the necessary hydrostatic/hydrodynamic information must be provided as input; it can be generated from the results of external BEM (Boundary Element Method) software with the BEMIO (Boundary Element Method Input/Output) functions of WEC-Sim. MOST can be used together with BEMIO and the external tools NEMOH [7] and SALOME to create the .h5 file automatically. To use this functionality, the user has to install SALOME 9.9.0 and the executable of NEMOH 3.0.0, then place these in the folder 'hydroDataMaker/NEMOH/EXE'. By running the 'h5Maker.m' script in the 'hydroDataMaker' folder, it is possible to create the .h5 file containing the hydrodynamic data of the floating body with a python script as input, which defines the actions performed in SALOME, as well as some settings to be adjusted in the 'h5Maker.m' script. To date, there are .py codes for the creation of Spar type and semi-submersible type (similar to the VolturnUS-S) platforms (it is sufficient to define the main dimensions in the 'h5Maker.m' scipt). However, to manipulate platforms with different shapes, it is enough to create a .py file similar to the existing ones while respecting some common inputs and outputs (see Table 13).

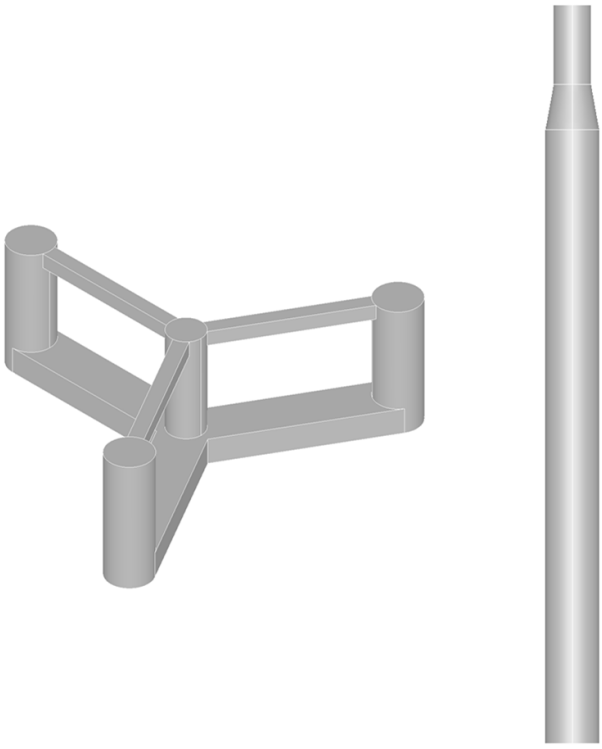

Figure 12: Platforms example

The following main steps are performed with the 'h5Maker.m' script:

- creation of geometric inputs for SALOME from settings
- SALOME execution, in which the geometry is created (from which the BEM software mesh

and the .stl file for visualisation are derived) and some inertial and hydrostatic properties are calculated

- post-processing SALOME output
- conversion of the mesh file into a format suitable to be read by NEMOH and creation of the necessary input files (see Figure 13)
- execution of NEMOH
- post-processing of NEMOH results using BEMIO functions and display of some results

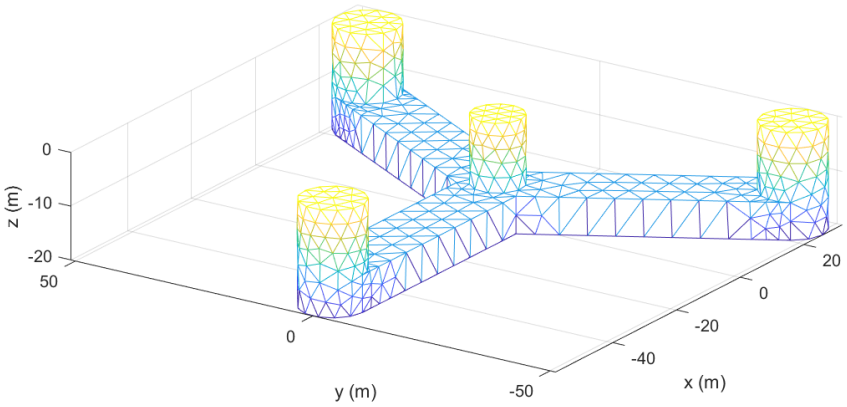

Figure 13: NEMOH mesh example

All necessary data are defined in the 'SETTING' section of the script 'h5Maker.m' and are contained in a structure whose main field is 'data'. The following table contains the names of the sub-fields (common for each type of floating body) and their description.

Name	Description
env.rho	water density (kg/m^3)

env.g	gravity acceleration (m/s^2)
env.depth	water depth
geom.Body_n.Maxsize_Nemoh	Approximate value of the maximum side length (m) of a mesh element (mesh file used in NEMOH)
geom.Body_n.Minsize_Nemoh	Approximate value of the maximum side length (m) of a mesh element (mesh file used in NEMOH)
geom.Body_n.Platform_Name	Name of the n-th body
geom.Body_n.PythonScriptName	name of the .py script used for n-th body
WTcomponents.Body_n.filepath	Path of the .mat file containing wind turbine data (created in pre-processing) mounted on the n-th body (if present)
mooring.Body_n.filepath	Path of the .mat file containing mooring system data (created in pre-processing) linked with the n-th body (if present)
salome.SALOME_path	Path of SALOME executable "run_SALOME.exe"
nemoh.frequencies	*1x3* row vector containing the number of frequency analyzed in NEMOH and range limits (the frequencies will be equi-spaced)
nemoh.wavedir	*1x3* row vector containing the number of wave direction analyzed in NEMOH and range limits (the directions will be equi-spaced)
nemoh.symmetric	flag to decide if exploit XZ-plane mesh symmetry
nemoh.results_folder_name	name of the folder where NEMOH results are stored
BEMIO	the variables in this subfield are the inputs of the BEMIO functions
path.WECSim_source	WEC-Sim/source folder path
flags.plot_hydro_result	flag for activation of BEMIO results plot
flags.plot_mesh_Nemoh	flag for activation of NEMOH mesh plot

Table 13: h5Maker.m inputs

In the sub-fields data.geom.Body_n, the geometric data to be supplied to SALOME must also be specified, according to the .py script. Below is the data supplied for the creation of the semi-submersible platform type (Figure14) already present in MOST examples.

Name	**Description**
D_MC	central column diameter (m)
D_EC	external columns diameter (m)
Draft	platform draft (m)
FB	platform freeboard (m)
L_EXT	distance between central and external columns (m)
W_ARM	pontoons width (m)

H_ARM	pontoons height (m)
W_BR	braces width (m)
H_BR	braces height (m)
t_MC	central column thickness (m)
t_EC	external columns thickness (m)
t_AR	pontoons thickness (m)
t_BR	braces thickness (m)

Table 14: Semi-submersible platform geometric inputs example

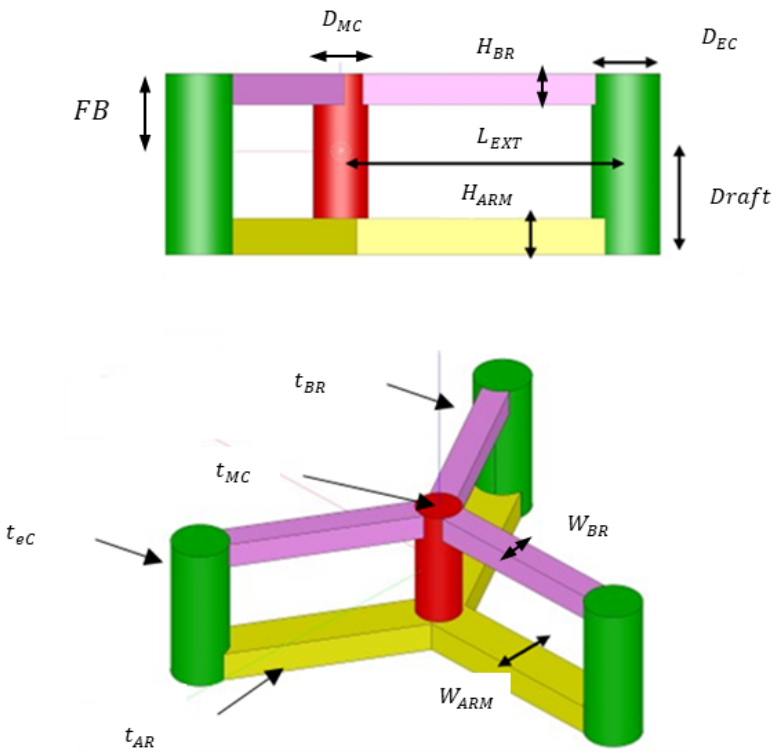

Figure 14: Floating body geometric inputs example

4.2 Simulation

WEC-Sim uses object-oriented programming based on certain classes (wave, body, moorings, etc.) that must be initialized by setting the main input file 'wecSimInputFile.m'. In addition to these, a series of Simulink functions and libraries complete what is needed to perform numerical simulations. MOST introduces the 'windClass' and 'windTurbineClass' classes and the

MOST_Lib.slx library, which includes a wind turbine model, into the WEC-Sim environment. The classes introduced serve to define the properties of wind and wind turbine, and with MOST, additional functionality is introduced within the "mooringClass" class and the "responseClass" (for the post-processing). For a detailed explanation of the classes and how to set the 'wecSimInputFile.m' script, please refer to the WEC-Sim documentation (WEC-Sim classes and Code Structure).

The following sections will explain only the classes and the library introduced/modificated with MOST. The classes are initialized and the value of their properties set according to the user's needs in the 'wecSimInputFile.m' file in the same way as for the classes already present in WEC-Sim. Once the file has been set, the simulation can be launched from the command line with the command "wecSim" or from directly from Simulink (refer to Running from Simulink).

windClass

The "windClass" creates a "wind" object saved to the MATLAB workspace. The "windClass" includes properties and methods used to define wind input.

The following table resumes the "SetAccess = 'public'" windClass properties.

Name	Description
ConstantWindFlag	Flag whose value determines the type of wind field used (0 means turbulent wind, while 1 means constant wind, see theory. Deafault: 0
V_time_breakpoints	Used for constant wind, a vector containing a series of time instants for which the direction and modulus of wind speed (constant in space) are established. Between two breakpoints, these quantities are obtained by linear interpolation. Default: [0 realmax]
V_dt	Number determining the temporal discretization of the wind field, only used for constant wind. Default: 0.1
V_modules	Vector of the same size as 'V_time_breakpoints' containing the values of the wind speed modules corresponding to those time instants, only used for constant wind. Default: [0 0]
V_directions	Used for constant wind, matrix *nx3* (where n is the length of the vector 'V_time_breakpoints'), each row is the vector whose direction corresponds to the wind direction at the time instants defined by 'V_time_breakpoints'. It is not necessary to define vectors of unit modulus. Default: [1 0 0;1 0 0]
V_domain_sizes	Vector *1x3* whose components are the dimensions along *x-y-z* of the box within which the wind field is defined, only used for constant wind. Default: [1e4 1e4 1e4]

WindDataFile	Path (relative to 'wecSimInputFile.m') to the file containing data of the turbulent wind field to be used in the simulation, only used for turbulent wind. For more information on how to create this file, see pre-processing section

Table 15: windClass properties

The following table lists the functions of the "winClass" methods.

Name	Description
ComputeWindInput	The function, based on the value of 'ConstantWindFlag', generates the wind field from the information contained in the properties. In the case of turbulent wind, it reads the provided file and uses its information to define the time history of the wind speed for each point of the spatial grid defined during its generation; in the case of constant wind, it creates the wind field from the modulus and direction values chosen by the user for each time breakpoints. Speed and direction are the same for each point in the spatial grid and vary linearly passing through the defined breakpoints characteristics

Table 16: windClass methods

windTurbineClass

The "windTurbineClass" creates a "windTurbine" object saved to the MATLAB workspace. The "windTurbineClass" includes properties and methods used to define wind turbine input.

The following table resumes the "SetAccess = 'public'" windTurbineClass properties.

Name	Description
name	"windTurbine" object name
turbineName	Path (relative to 'wecSimInputFile.m') to the file containing the wind turbine geometric and inertial properties created here
bladeDataName	Path (relative to 'wecSimInputFile.m') to the file containing the wind turbine's blades properties created here
offset_plane	Two component row vector containing the *x*-*y* coordinates where the tower center is positioned during the simulation. Default: [0 0]
control	Flag to choose control type, 0: Baseline, 1:ROSCO, for more information about controls logic see the theory. Default: 0

controlName	Path (relative to 'wecSimInputFile.m') to the file containing the controller data created here
YawControlFlag	Flag to activate (1) or disactivate (0) the yaw control. If 0 is chosen, the rotational degree of freedom between tower and nacelle is locked. Default: 0
omega0	Initial value of rotor angular speed. Default: empty, in this case the rated condition value is set (depending on the controller chosen)
bladepitch0	Initial value of blade collective pitch. Default: empty, in this case the rated condition value is set (depending on the controller chosen)
GenTorque0	Initial value of generator torque. Default: empty, in this case the rated condition value is set (depending on the controller chosen)
aeroLoadsType	Flag to choose the aerodynamic loads computation method, 0: look-up tables, 1: Blade Element Momentum problem resolution at each time step, see pre-processing section. Default: 0
aeroLoadsName	Path (relative to 'wecSimInputFile.m') to the file containing the aerodynamic loads look-up tables, only used for aeroLoadsType=0. For more information on how to create this file, see pre-processing section
BEMdata	Structure containing some data to solve the Blade Element Momentum problem during the simulation, only used for aeroLoadsType=1. They are the ari density and some instructions that determine how the iterative resolution process has to converge. Moreover, here the user can specify (if needed) how to decimate the aerodynamic nodes along the blades, if one wants to reduce the computational time (but with less precision). Please consult the 'CreateBEMstruct' function and the portion of the wind turbine Simulink model dedicated to calculating aerodynamic loads to better understand the function of these data

Table 17: windTurbineClass properties

The following table lists the functions of the "windTurbineClass" methods.

Name	Description
importAeroLoadsTable	The function (used for aeroLoadsType=0) loads the file containing the look-up tables of aerodynamic loads. They will be different depending on the choice of controller

importControl	The function loads the file containing the controllers data and (depending on which controller is chosen with the flag "control") assigns initial values of rotor speed, generator torque and blade pitch (rated values) if they are not provided by user (properites "omega0", "bladepitch0" and "GenTorque0" of "windTurbineClass")
loadTurbineData	The function loads the files containing the geometric and inertial wind turbine data and the blades data created during pre-processing section
CreateBEMstruct	The function (used for aeroLoadsType=1) set the "BEMstruct" structure used in the Simulink model to compute the aerodynamics loads, depending on "BEMdata" settings and on "wind" object features

Table 18: windTurbineClass methods

mooringClass

MOST introduces the possibility of simulating the mooring system with non-linear look-up tables previously calculated in the pre-processing phase by solving the static catenary equations or by solving these equations at each time step via optimization procedures (see theory for more information). For the first case, the 'MooringLookupTable' sub-model of the 'WEC-Sim_Lib_Moorings.slx' library will be used, while for the second case, the 'MooringNLStatic' sub-model of the same library will be used. The properties and methods of the 'mooringClass' introduced with MOST will be explained here, please consult the WEC-Sim documentation for complete information on the 'mooringClass'.

The following table resumes the "SetAccess = 'public'" mooringClass properties introduced with MOST.

Name	Description
lookupTableFile	Mooring look-up table file name, created here
Data_moor	Structure containing the information needed to simulate the mooring system, such as the number of lines, diameter, linear mass density, etc. This information is the same as that provided in the pre-processing phase for the creation of look-up tables, for more information see the dedicated section

Table 19: mooringClass properties

The following table lists the functions of the "mooringClass" methods introduced with MOST.

Name	Description

loadLookupTable	The function (used when mooring system is modeled through look-up tables) loads the file containing the look-up tables of mooring loads
NLStatic_Setup	The function (used when mooring system loads are computed through 'MooringNLStatic' sub-model) set the "BEMstruct" structure used in the Simulink model to compute the mooring loads

Table 20: mooringClass methods

MOST_Lib

In this section, an overview is given of the MOST_Lib.slx library (in 'source/lib/MOST' folder), i.e. the Simulink model representing the wind turbine (Figure 15). It is constructed exploiting Simscape Multibody blocks and, to summarize, consists of a series of blocks through which the masses and inertia of the components are modeled, as well as the part concerning the control and the aerodynamic loads computation.

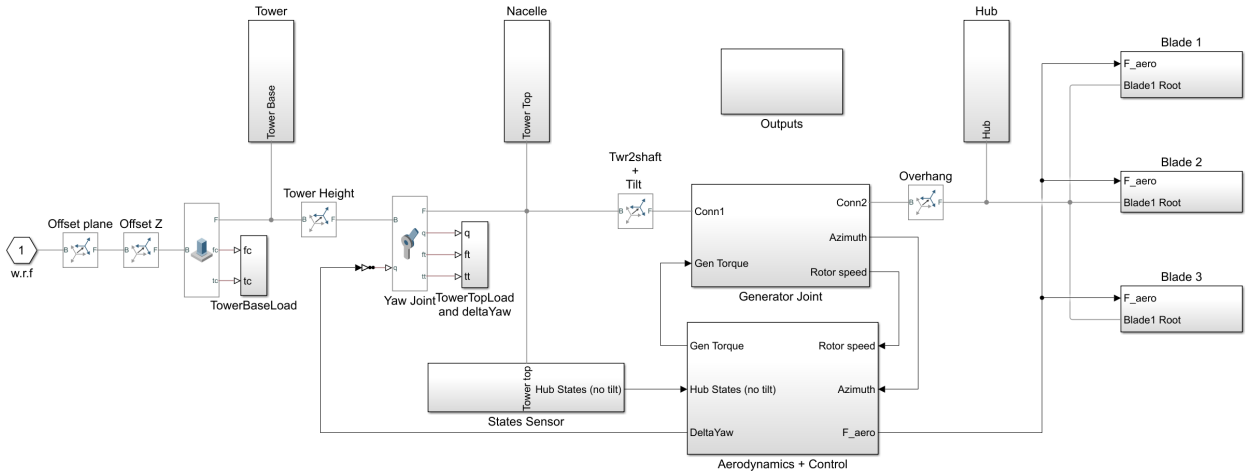

Figure 15: Wind turbine Simulink library

Specifically, the turbine consists of six components: tower, nacelle, hub and three blades. For each of these components, a File Solid block is used to describe its geometry (for visualization only), mass and inertia. These are linked together using Weld Joint and Revolute Joint blocks, the latter to model the rotation between tower and nacelle and between nacelle and hub. The weight force is applied to each component using External Force and Torque block and, in the case of the blades, aerodynamic loads are applied to their roots. Figure 16 shows, as an example, the sub-model for the tower component.

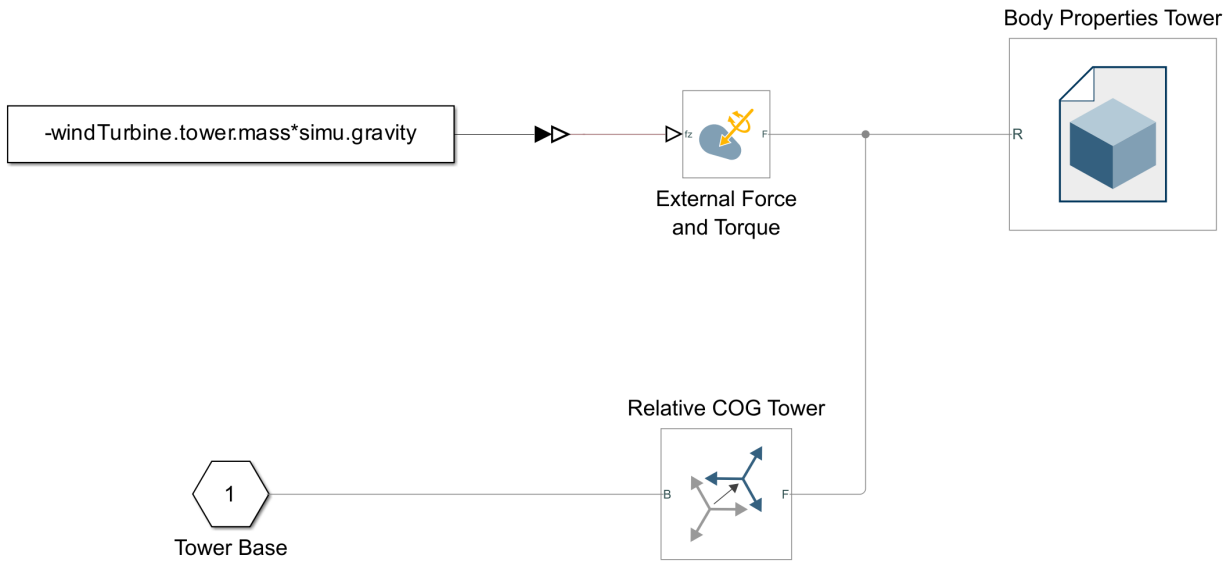

Figure 16: Tower Simulink sub-model

In addition to the aforementioned blocks, a number of Rigid Transform blocks are used for the roto-translation of local reference systems and Transform Sensor blocks are used to derive some useful kinematic quantities, e.g. for the calculation of aerodynamic forces or used as input in the control sub-models. In the sub-model 'Aerodynamics + Control' (Figure 17 there are three further sub-models dedicated to power control, yaw control and calculation of aerodynamic loads. With regard to power control, there is a Variant Subsystem and depending on the user's choice, ROSCO or Baseline control logic is applied. Similarly, with regard to the calculation of aerodynamic loads, there is a Variant Subsystem and depending on the user's choice, look-up tables are used or the BEM problem is solved at each time step.

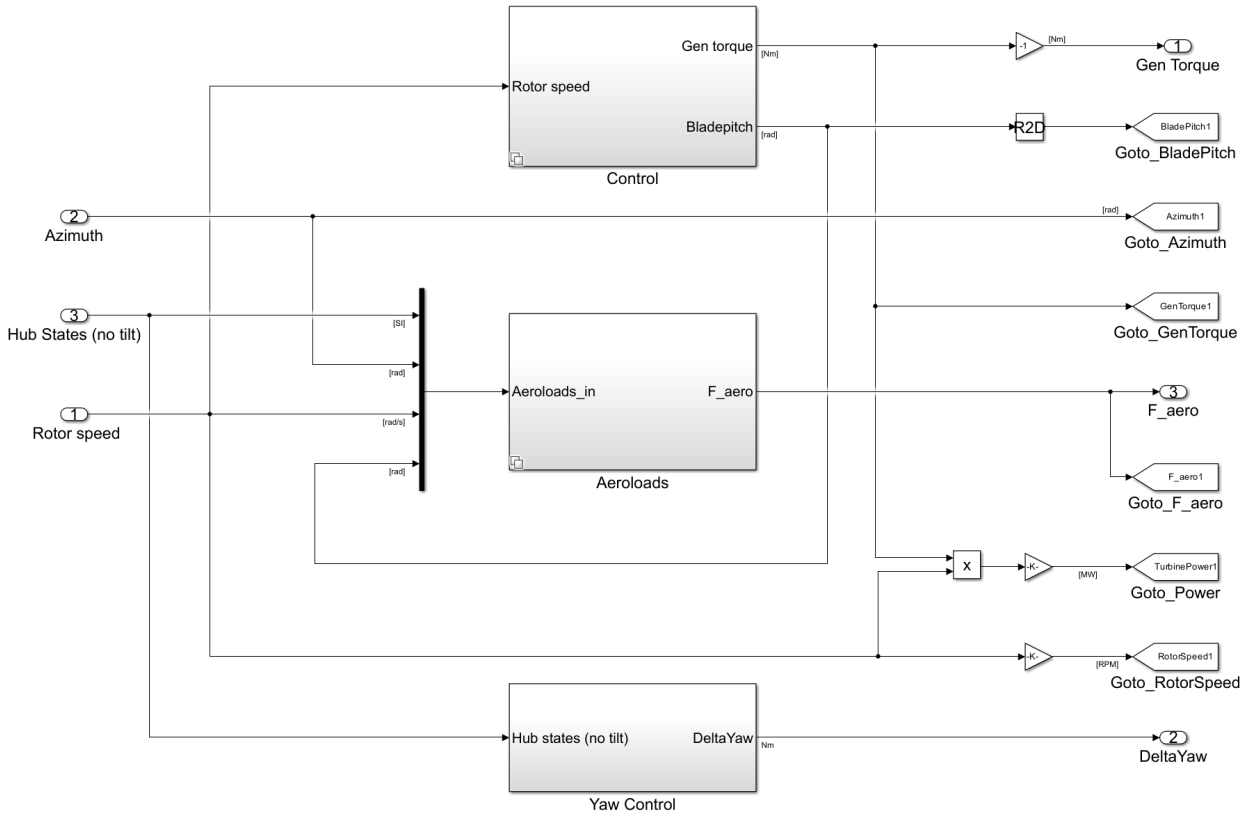

Figure 17: Aerodynamics + Control Simulink sub-model

4.3 Post-processing

Once the simulation is complete, the 'stopWecSim.m' and 'postProcessWecSim.m' scripts are launched via the 'wecSim.m' script. In these, a number of operations are performed, including generating the 'output' object of the rensponseClass class, in which the optimization outputs are arranged. Depending on the value of certain properties of the simulationClass, it is also possible to choose whether to save the workspace and whether to create summary text files of the results. Furthermore, if the script 'wecSimInputFile.m' exists in the folder in which the script 'userDefinedFunctions.m' exists, it is launched. This is a script created by the user so usually used to display the optimization results (see MOST examples). For more information on how the outputs in the 'rensponseClass' class are handled, please refer to the WEC-Sim documentation, here we simply list the outputs introduced with MOST, i.e. those related to wind and wind turbine.

Below is the table containing the outputs' names for the wind turbine and wind field that are saved in the "output" object.

Name	Description
windSpeed	*time-stepsx3*, wind speed vector at hub rest position
turbinePower	*time-stepsx1*, wind turbine extracted power

rotorSpeed	*time-stepsx1*, rotor angular speed
bladePitch	*time-stepsx1*, collective blade pitch
nacelleAcceleration	*time-stepsx3*, nacelle acceleration vector
NacXdot	*time-stepsx1*, nacelle velocity along x-axis (world reference frame)
towerBaseLoad	*time-stepsx6*, loads at the weld joint at the tower base
towerTopLoad	*time-stepsx6*, loads at the weld joint between tower and nacelle
blade1RootLoad	*time-stepsx6*, loads at the weld joint between hub and blade 1, located at blade root
genTorque	*time-stepsx1*, generator torque
azimuth	*time-stepsx1*, rotor azimuthal angle
blade1AeroLoad	*time-stepsx6*, aeroloads at the blade 1 root
blade2AeroLoad	*time-stepsx6*, aeroloads at the blade 2 root
blade3AeroLoad	*time-stepsx6*, aeroloads at the blade 3 root
DeltaYaw	*time-stepsx1*, relative angle between tower and nacelle

Table 21: wind turbine outputs

The Figure 18 below illustrates sample input-output plots from a simulation of the IEA 15 MW reference wind turbine on the VolturnUS semi-submersible platform.

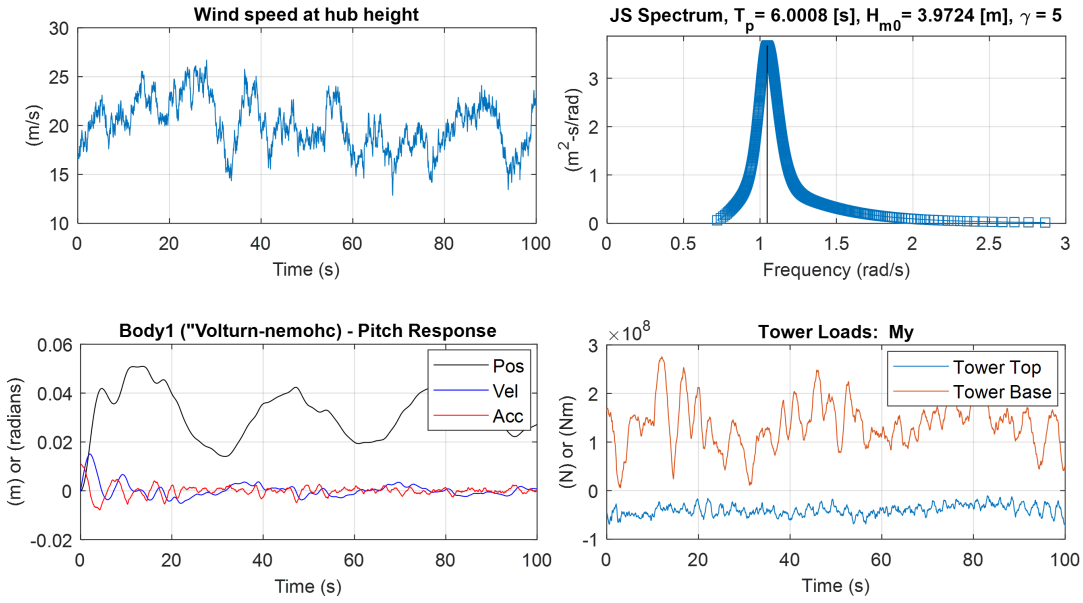

Figure 18: Example of results simulating a floating wind turbine (IEA 15 MW WT).

References

[1] N. J. Abbas, D. S. Zalkind, L. Pao, and A. Wright. A reference open-source controller for fixed and floating offshore wind turbines. *Wind Energy Science*, 7(1):53–73, 2022. doi: 10.5194/wes-7-53-2022. URL https://wes.copernicus.org/articles/7/53/2022/.

[2] Christopher Allen, Anthony Viscelli, Habib Dagher, Andrew Goupee, Evan Gaertner, Nikhar Abbas, Matthew Hall, and Garrett Barter. Definition of the umaine volturnus-s reference platform developed for the iea wind 15-megawatt offshore reference wind turbine. 7 2020. doi: 10.2172/1660012. URL https://www.osti.gov/biblio/1660012.

[3] Evan Gaertner, Jennifer Rinker, Latha Sethuraman, Frederik Zahle, Benjamin Anderson, Garrett E Barter, Nikhar J Abbas, Fanzhong Meng, Pietro Bortolotti, Witold Skrzypinski, George N Scott, Roland Feil, Henrik Bredmose, Katherine Dykes, Matthew Shields, Christopher Allen, and Anthony Viselli. Iea wind tcp task 37: Definition of the iea 15-megawatt offshore reference wind turbine. 3 2020. doi: 10.2172/1603478. URL https://www.osti.gov/biblio/1603478.

[4] M.H. Hansen, Anca Daniela Hansen, Torben J. Larsen, S. Øye, P. Sørensen, and P. Fuglsang. *Control design for a pitch-regulated, variable speed wind turbine*. Number 1500(EN) in Denmark. Forskningscenter Risoe. Risoe-R. 2005. ISBN 87-550-3409-8.

[5] B J Jonkman and Jr. Buhl, M. L. Turbsim user's guide. 9 2006. doi: 10.2172/891594. URL https://www.osti.gov/biblio/891594.

[6] J Jonkman, S Butterfield, W Musial, and G Scott. Definition of a 5-mw reference wind turbine for offshore system development. 2 2009. doi: 10.2172/947422. URL https://www.osti.gov/biblio/947422.

[7] Ruddy Kurnia and Guillaume Ducrozet. Nemoh: Open-source boundary element solver for computation of first- and second-order hydrodynamic loads in the frequency domain. *Computer Physics Communications*, 292:108885, 2023. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2023.108885. URL https://www.sciencedirect.com/science/article/pii/S0010465523002308.

[8] Andrew Ning, Gregory Hayman, Rick Damiani, and Jason M. Jonkman. *Development and Validation of a New Blade Element Momentum Skewed-Wake Model within AeroDyn*. doi: 10.2514/6.2015-0215. URL https://arc.aiaa.org/doi/abs/10.2514/6.2015-0215.

[9] S. Andrew Ning. A simple solution method for the blade element momentum equations with guaranteed convergence. *Wind Energy*, 17(9):1327–1345, 2014. doi: https://doi.org/10.1002/we.1636. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1636.