ROSCO toolbox

Release v2.2.0

ROSCO developers

Jun 11, 2021

CONTENTS

NREL's Reference OpenSource Controller (ROSCO) toolbox for wind turbine applications is a toolbox designed to ease controller implementation for the wind turbine researcher. The purpose of these documents is to provide information for the use of the ROSCO related toolchain.

Figure Fig. 1 shows the general workflow for the ROSCO toolchain.

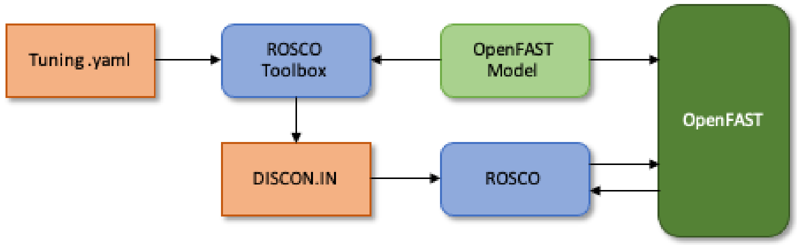

Fig. 1: ROSCO toolchain general workflow

ROSCO Toolbox

- Generic tuning of NREL's ROSCO controller
- Simple 1-DOF turbine simulations for quick controller capability verifications
- Parsing of OpenFAST input and output files
- Block diagrams of these capabilities can be seen in architecture.png.

ROSCO Controller

- Fortran based
- Follows Bladed-style control interface
- Modular

CHAPTER

ONE

STANDARD USE

For the standard use case in OpenFAST, ROSCO will need to be compiled. This is made possible via the instructions found in *Installing the ROSCO tools*. Once the controller is compiled, the turbine model needs to point to the compiled binary. In OpenFAST, this is ensured by changing the `DLL_FileName` parameter in the ServoDyn input file.

Additionally, an additional input file is needed for the ROSCO controller. Though the controller only needs to be compiled once, each individual turbine/controller tuning requires an input file. This input file is generically dubbed "DISCON.IN". In OpenFAST, the `DLL_InFile` parameter should be set to point to the desired input file. The ROSCO toolbox is used to automatically generate the input file. These instructions are provided in the instructions for *Standard ROSCO Workflow*.

CHAPTER

TWO

TECHNICAL DOCUMENTATION

A publication highlighting much of the theory behind the controller tuning and implementation methods can be found at: https://wes.copernicus.org/preprints/wes-2021-19/

CHAPTER

THREE

SURVEY

Please help us better understand the ROSCO user-base and how we can improve ROSCO through this brief survey:

CHAPTER

FOUR

DIRECTORY

4.1 Installing the ROSCO tools

Depending on what is needed, a user can choose to use just the ROSCO controller or to use both the ROSCO controller and the toolbox. Both the controller and the toolbox should be installed if one wishes to leverage the full ROSCO toolchain.

For users who wish to use the ROSCO toolbox (with or without the controller), please skip to the section on section *ROSCO Toolbox Structure*. For users planning to only download and compile the ROSCO controller, please follow the instructions on *ROSCO controller*. For information on best practices to update to the most recent version of the ROSCO toolbox, see *Updating the ROSCO Toolbox*.

4.1.1 ROSCO controller

The standard ROSCO controller is based in Fortran and must be compiled; this code can be found at: https://github.com/NREL/ROSCO. Of course, the advanced user can compile the downloaded code using their own desired methods (e.g. Visual Studio). Otherwise, a few of the more common compiling methods are detailed on this page. Additionally, the most recent tagged version releases are available for download.

If one wishes to download the code via the command line, we provide two supported options in the subsections below. For non-developers (those not interested in modifying the source code), the a 64-bit version of the compiled controller can be downloaded via Anaconda. For users needing a 32-bit version on Windows and/or developers, CMake can be used to compile the Fortran code.

Anaconda download for non-developers

For users familiar with Anaconda, a 64-bit version of ROSCO is available through the conda-forge channel. In order to download the most recently compiled version release, from an anaconda powershell (Windows) or terminal (Mac/Linux) window, create a new anaconda virtual environment:

```
conda config --add channels conda-forge
conda create -y --name rosco-env python=3.8
conda activate rosco-env
```

navigate to your desired folder to save the compiled binary using:

```
cd <my_desired_folder>
```

and download the controller:

```
conda install -y ROSCO
```

This will download a compiled ROSCO binary file into the default filepath for any dynamic libraries downloaded via anaconda while in the ROSCO-env. The ROSCO binary file can be copied to your desired folder using:

```
cp $CONDA_PREFIX/lib/libdiscon.* .
```

on linux or:

```
copy %CONDA_PREFIX%/lib/libdiscon.* .
```

on Windows.

CMake for developers (Mac/linux)

CMake provides a straightforward option for many users, particularly those on a Mac or Linux. On Mac/Linux, ROSCO can be compiled by first cloning the source code from git using:

```
git clone https://github.com/NREL/ROSCO.git
```

And then compiling using CMake:

```
cd ROSCO
mkdir build
cd build
cmake ..
make install
```

This will generate a file called `libdiscon.so` (Linux) or `libdiscon.dylib` (Mac) in the `/ROSCO/install/lib` directory.

CMake for developers/32-bit (Windows)

To compile ROSCO on Windows, you first need a Fortran compiler. If you need a 32-bit DLL, then we recommend installing MinGW (Section 2). If you require a 64-bit version, you can install the MSYS2 toolchain through conda:

```
conda install m2w64-toolchain libpython
```

Note that if you have the 64-bit toolchain installed in your environment, you might have conflicts with the 32-bit compiler. We recommend therefore keeping separate environments if you want to compile 32- or 64-bit.

Once you have your Fortran compiler successfully installed and configured, the build process is similar to on Mac and linux:

```
cd ROSCO
mkdir build
cd build
cmake .. -G "MinGW Makefiles"
mingw32-make
```

Note that the `mingw32-make` command is (confusingly) valid for both 64-bit and 32-bit MinGW.

This will generate a file called `libdiscon.dll` in the `/ROSCO/install/lib` directory.

4.1.2 Full ROSCO toolbox

We recommend using the full ROSCO toolbox so that you can leverage the entire toolchain.

Installing

Installation of the complete ROSCO toolbox is made easy through Anaconda. If you do not already have Anaconda installed on your machine, please install it.

Then please follow the following steps:

1. Create a conda environment for ROSCO

```
conda config --add channels conda-forge
conda create -y --name rosco-env python=3.8
conda activate rosco-env
```

2. Install WISDEM

```
conda install -y wisdem
```

You should then do step three *or* four. If you do not want to compile the ROSCO controller within the installation of the ROSCO toolbox, please follow the instructions for compiling_rosco.

3. Clone and Install the ROSCO toolbox with ROSCO

```
git clone https://github.com/NREL/ROSCO_toolbox.git
cd ROSCO_toolbox
git submodule init
git submodule update
conda install compilers                                         # (Mac/Linux only)
conda install m2w64-toolchain libpython     # (Windows only)
python setup.py install --compile-rosco
```

4. Clone and Install the ROSCO toolbox *without* ROSCO

```
git clone https://github.com/NREL/ROSCO_toolbox.git
cd ROSCO_toolbox
python setup.py install
```

Alternatively...

If you wish to write your own scripts to leverage the ROSCO toolbox tools, but do not necessarily need the source code or to run any of the examples, the ROSCO toolbox is available via PyPi:

```
pip install rosco_toolbox
```

Note that if you do choose to install the ROSCO Toolbox this way, you will not have the source code. Additionally, you will need to download WISDEM and the ROSCO controller separately if you wish to use any of the ROSCO toolbox functionalities that need those software packages.

Updating the ROSCO Toolbox

Simple git commands should update the toolbox and controller as development continues: ` git pull git submodule update ` and then recompile and reinstall as necessary...

Getting Started

Please see a the *Standard ROSCO Workflow* for several example scripts using ROSCO and the ROSCO_toolbox.

4.2 Standard ROSCO Workflow

This page outlines methods for reading turbine models, generating the control parameters of a `DISCON.IN`: file, and running aeroelastic simulations to test controllers. A set of example scripts demonstrate the functionality of ROSCO_toolbox and ROSCO controller.

4.2.1 Reading Turbine Models

Control parameters depend on the turbine model. The ROSCO_toolbox uses OpenFAST inputs and an additional `.yaml` formatted file to set up a `turbine` object in python. Several OpenFAST inputs are located in Test_Cases/. The controller tuning `.yaml` are located in Tune_Cases/. A detailed description of the ROSCO control inputs and tuning `.yaml` are provided in *The DISCON.IN file* and *The ROSCO Toolbox Tuning File*, respectively.

- `example_01.py` loads an OpenFAST turbine model and displays a summary of its information
- `example_02.py` plots the C_p surface of a turbine

ROSCO requires the power and thrust coefficients for tuning control inputs and running the extended Kalman filter wind speed estimator.

- `example_03.py` runs cc-blade, a blade element momentum solver from WISDEM, to generate a C_p surface.

The `Cp_Cq_Ct.txt` (or similar) file contains the rotor performance tables that are necessary to run the ROSCO controller. This file can be located wherever you desire, just be sure to point to it properly with the `PerfFileName` parameter in `DISCON.IN`.

4.2.2 Tuning Controllers and Generating DISCON.IN

The ROSCO `turbine` object, which contains turbine information required for controller tuning, along with control parameters in the tuning yaml and the C_p surface are used to generate control parameters and `DISCON.IN` files. To tune the PI gains of the torque control, set `omega_vs` and `zeta_vs` in the yaml. Similarly, set `omega_pc` and `zeta_pc` to tune the PI pitch controller; gain scheduling is automatically handled using turbine information. Generally `omega_*` increases the responsiveness of the controller, reducing generator speed variations, but an also increases loading on the turbine. `zeta_*` changes the damping of the controller and is generally less important of a tuning parameter, but could also help with loading. The default parameters in Tune_Cases/ are known to work well with the turbines in this repository.

- `example_04.py` loads a turbine and tunes the PI control gains
- `example_05.py` tunes a controller and runs a simple simualtion (not using OpenFAST)
- `example_06.py` loads a turbine, tunes a controller, and runs an OpenFAST simulation

Each of these examples generates a `DISCON.IN` file, which is an input to libdiscon.*. When running the controller in OpenFAST, `DISCON.IN` must be appropriately named using the `DLL_FileName` parameter in ServoDyn.

OpenFAST can be installed from source or in a conda environment using:

```
conda install -c conda-forge openfast
```

ROSCO can implement peak shaving (or thrust clipping) by changing the minimum pitch angle based on the estimated wind speed:

- `example_07.py` loads a turbine and tunes a controller with peak shaving.

By setting the `ps_percent` value in the tuning yaml, the minimum pitch versus wind speed table changes and is updated in the `DISCON.IN` file.

ROSCO also contains a method for distributed aerodynamic control (e.g., via trailing edge flaps):

- `example_10.py` tunes a controller for distributed aerodynamic control

4.2.3 Running OpenFAST Simulations

To run an aeroelastic simulation with ROSCO, the ROSCO input (`DISCON.IN`) must point to a properly formatted `Cp_Cq_Ct.txt` file using the `PerfFileName` parameter. If called from OpenFAST, the main OpenFAST input points to the ServoDyn input, which points to the `DISCON.IN` file and the `libdiscon.*` dynamic library.

For example in *Test_Cases/NREL-5MW*:

- `NREL-5MW.fst` has `"NRELOffshrBsline5MW_Onshore_ServoDyn.dat"` as the `ServoFile` input
- `NRELOffshrBsline5MW_Onshore_ServoDyn.dat` has `"../../ROSCO/build/libdiscon.dylib"` as the `DLL_FileName` input and `"DISCON.IN"` as the `DLL_InFile` input. Note that these file paths are relative to the path of the main fast input (`NREL-5MW.fst`)
- `DISCON.IN` has `"Cp_Ct_Cq.NREL5MW.txt"` as the `PerfFileName` input

The ROSCO_toolbox has methods for running OpenFAST (and other) binary executables using system calls, as well as post-processing tools in ofTools/.

Several example scripts are set up to quickly simulate ROSCO with OpenFAST:

- `example_06.py` loads a turbine, tunes a controller, and runs an OpenFAST simulation
- `example_08.py` loads the OpenFAST output files and plots the results
- `example_09.py` runs TurbSim, for generating turbulent wind inputs

4.2.4 Testing ROSCO

The ROSCO_toolbox also contains tools for testing ROSCO in IEC design load cases (DLCs), located in ROSCO_testing/. The script `run_Testing.py` allows the user to set up their own set of tests. By setting `testtype`, the user can run a variety of tests:

- `lite`, which runs DLC 1.1 simulations at 5 wind speed from cut-in to cut-out, in 330 second simulations
- `heavy`, which runs DLC 1.3 from cut-in to cut-out in 2 m/s steps and 2 seeds for each, in 630 seconds, as well as DLC 1.4 simulations
- `binary-comp`, where the user can compare `libdiscon.*` dynamic libraries (compiled ROSCO source code), with either a lite or heavy set of simulations

- discon-comp, where the user can compare DISCON.IN controller tunings (and the complied ROSCO source is constant)

Setting the turbine2test allows the user to test either the IEA-15MW with the UMaine floating semisubmersible or the NREL-5MW reference onshore turbine.

4.3 ROSCO Toolbox Structure

Here, we give an overview of the structure of the ROSCO toolbox and how the code is implemented.

4.3.1 File Structure

The primary tools of the ROSCO toolbox are separated into several folders. They include the following:

ROSCO_toolbox

The source code for the ROSCO toolbox generic tuning implementations lives here.

- turbine.py loads a wind turbine model from OpenFAST input files.
- controller.py contains the generic controller tuning scripts
- utilities.py has most of the input/output file management scripts
- control_interface.py enables a python interface to the ROSCO controller
- sim.py is a simple 1-DOF model simulator
- **ofTools** is a folder containing a large set of tools to handle OpenFAST input files - this is primarily used to run large simulation sets and to handle reading and processing of OpenFAST input and output files.

Examples

A number of examples are included to showcase the numerous capabilities of the ROSCO toolbox; they are described in the *Standard ROSCO Workflow*.

Matlab_Toolbox

A simulink implementation of the ROSCO controller is included in the Matlab Toolbox. Some requisite MATLAB utility scripts are also included.

ROSCO_testing

Testing scripts for the ROSCO toolbox are held here and showcased with run_testing.py. These can be used to compare different controller tunings or different controllers all together.

Test_Cases

Example OpenFAST models consistent with the latest release of OpenFAST are provided here for simple testing and simulation cases.

Tune_Cases

Some example tuning scripts and tuning input files are provided here. The code found in `tune_ROSCO.py` can be modified by the user to easily enable tuning of their own wind turbine model.

4.3.2 The ROSCO Toolbox Tuning File

A yaml formatted input file is used for the standard ROSCO toolbox tuning process. This file contains the necessary inputs for the ROSCO toolbox to load an OpenFAST input file deck and tune the ROSCO controller. It contains the following inputs:

Table 4.1: ROSCO toolbox input yaml

Primary Section	Variable	Required	Type	Description
`path_params`	`FAST_InputFile`	Yes	String	Name of the primary (*.fst) OpenFAST input file
	`FAST_directory`	Yes	String	Main OpenFAST model directory, where the *.fst lives
	`rotor_performance_filename`	No	String	Filename for rotor performance text file. If this is not specified, and an existing rotor performance file cannot be found, cc-blade will be run
`turbine_params`	`rotor_interia`	Yes	Float	Rotor inertia [kg m^2], (Available in Elastodyn .sum file)
	`rated_rotor_speed`	Yes	Float	Rated rotor speed of the turbine [rad/s]
	`v_min`	Yes	Float	Cut-in wind speed [m/s]
	`v_max`	Yes	Float	Cut-out wind speed [m/s]
	`max_pitch_rate`	Yes	Float	Maximum blade pitch rate [rad/s]
	`max_torque_rate`	Yes	Float	Maximum generator torque rate [Nm/s]
	`rated_power`	Yes	Float	Rated Power [W].
	`bld_edgewise_freq`	Yes	Float	Blade edgewise first natural frequency [rad/s]. Set this even if you are using stiff blades. It becomes the generator speed LPF bandwidth.
	`TSR_operational`	No	Float	Desired below-rated operation tip speed ratio [-]. If this is not specified, the Cp-maximizing TSR from the Cp surface is used.
	`twr_freq`	No	Float	Tower first fore-aft natural frequency [rad/s]. Required for floating wind turbine control.
	`ptfm_freq`	No	Float	Platform first fore-aft natural frequency [rad/s]. Required for floating wind turbine control.
`controller_params`	`LoggingLevel`	Yes	Int	0: write no debug files, 1: write standard output .dbg-file, 2: write standard output .dbg-file and complete avrSWAP-array .dbg2-file
	`F_LPFType`	Yes	Int	Type of Low pass filter for the generator speed feedback signal [rad/s]. 1: first-order low-pass filter, 2: second-order low-pass filter.

continues on next page

Table 4.1 – continued from previous page

Primary Section	Variable	Required	Type	Description
	F_NotchType	Yes	Int	Notch filter on generator speed and/or tower fore-aft motion, used for floating wind turbine control. 0: disable, 1: generator speed, 2: tower-top fore-aft motion, 3: generator speed and tower-top fore-aft motion
	IPC_ControlMode	Yes	Int	Turn Individual Pitch Control (IPC) for fatigue load reductions (pitch contribution). 0: off, 1: 1P reductions, 2: 1P+2P reductions.
	VS_ControlMode	Yes	Int	Generator torque control mode. 0: $k\omega^2$ below rated, constant torque above rated, 1: $k\omega^2$ below rated, constant power above rated, 2: TSR tracking PI control below rated, constant torque above rated, 3: TSR tracking PI control below rated, constant power above rated.
	PC_ControlMode	Yes	Int	Blade pitch control mode. 0: No pitch control, fix to fine pitch, 1: active PI blade pitch control
	Y_ControlMode	Yes	Int	Yaw control mode. 0: no yaw control, 1: yaw rate control, 2: yaw-by-IPC
	SS_Mode	Yes	Int	Setpoint Smoother mode. 0: no set point smoothing, 1: set point smoothing
	WE_Mode	Yes	Int	Wind speed estimator mode. 0: One-second low pass filtered hub height wind speed, 1: Immersion and Invariance Estimator (Ortega et al.), 2: Extended Kalman filter
	PS_Mode	Yes	Int	Pitch saturation mode. 0: no pitch saturation, 1: peak shaving, 2: Cp-maximizing pitch saturation, 3: peak shaving and Cp-maximizing pitch saturation
	SD_Mode	Yes	Int	Shutdown mode. 0: no shutdown procedure, 1: pitch to max pitch at shutdown.
	Fl_Mode	Yes	Int	Floating feedback mode. 0: no nacelle rotational velocity feedback, 1: nacelle rotational velocity feedback
	Flp_Mode	Yes	Int	Flap control mode. 0: no flap control, 1: steady state flap angle, 2: Proportional flap control
	zeta_pc	Yes	Float	Pitch controller desired damping ratio [-]
	omega_pc	Yes	Float	Pitch controller desired natural frequency [rad/s]
	zeta_vs	Yes	Float	Torque controller desired damping ratio [-]
	omega_vs	Yes	Float	Torque controller desired natural frequency [rad/s]
	zeta_flp	No	Float	Flap controller desired damping ratio [-]. Required if `Flp_Mode>0`
	omega_flp	No	Float	Flap controller desired natural frequency [rad/s]. Required if `Flp_Mode>0`
	max_pitch	No	Float	Maximum blade pitch angle [rad]. Default is 1.57 rad (90 degrees).
	min_pitch	No	Float	Minimum blade pitch angle [rad]. Default is 0 degrees.
	vs_minspd	No	Float	Minimum rotor speed [rad/s]. Default is 0 rad/s.
	ss_cornerfreq	No	Float	First order low-pass filter cornering frequency for setpoint smoother [rad/s]. Default is .6283 rad/s.
	ss_vsgain	No	Float	Torque controller set point smoother gain bias percentage [$\leq$ 1]. Default is 1.
	ss_pcgain	No	Float	Pitch controller set point smoother gain bias percentage [$\leq$ 1]. Default is 0.001.

continues on next page

Table 4.1 – continued from previous page

Primary Section	Variable	Required	Type	Description
	`ps_percent`	No	Float	Percent peak shaving [$\leq$ 1]. Default is 0.8.
	`sd_maxpit`	No	Float	Maximum blade pitch angle to initiate shutdown [rad]. Default is the blade pitch angle at `v_max`.
	`sd_cornerfreq`	No	Float	Cutoff Frequency for first order low-pass filter for blade pitch angle [rad/s]. Default is 0.41888 rad/s.
	`flp_maxpit`	No	Float	Maximum (and minimum) flap pitch angle [rad]. Default is 0.1745 rad (10 degrees).

4.4 ROSCO Controller Structure

Here, we give an overview of the structure of the ROSCO controller and how the code is implemented.

4.4.1 File Structure

The primary functions of the ROSCO toolbox are separated into several files. They include the following:

- `DISCON.f90` is the primary driver function.
- `ReadSetParameters.f90` primarily handles file I/O and the Bladed Interface.
- `ROSCO_Types.f90` allocates variables in memory.
- `Constants.f90` establishes some global constants.
- `Controllers.f90` contains the primary controller algorithms (e.g. blade pitch control)
- `ControllerBlocks.f90` contains additional control features that are not necessarily primary controllers (e.g. wind speed estimator)
- `Filters.f90` contains the various filter implementations.
- `Functions.f90` contains various functions used in the controller.

4.4.2 The DISCON.IN file

A standard file structure is used as an input to the ROSCO controller. This is, generically, dubbed the DISCON.IN file, though it can be renamed (In OpenFAST, this file is pointed to by `DLL_InFile` in the ServoDyn file. Examples of the DISCON.IN file are found in each of the Test Cases in the ROSCO toolbox, and in the `parameter_files` folder of ROSCO.

Table 4.2: DISCON.IN

Primary Section	Variable	Type	Description
DEBUG	`LoggingLevel`	Int	0: write no debug files, 1: write standard output .dbg-file, 2: write standard output .dbg-file and complete avrSWAP-array .dbg2-file

continues on next page

Table 4.2 – continued from previous page

Primary Section	Variable	Type	Description
CONTROLLER FLAGS	F_LPFType	Int	Filter type for generator speed feedback signal. 1: first-order low-pass filter, 2: second-order low-pass filter.
	F_NotchType	Int	Notch filter on the measured generator speed and/or tower fore-aft motion (used for floating). 0: disable, 1: generator speed, 2: tower-top fore-aft motion, 3: generator speed and tower-top fore-aft motion.
	IPC_ControlMode	Int	Individual Pitch Control (IPC) type for fatigue load reductions (pitch contribution). 0: off, 1: 1P reductions, 2: 1P+2P reductions.
	VS_ControlMode	Int	Generator torque control mode type. 0: $k\omega^2$ below rated, constant torque above rated, 1: $k\omega^2$ below rated, constant power above rated, 2: TSR tracking PI control below rated, constant torque above rated, 3: TSR tracking PI control below rated, constant torque above rated
	PC_ControlMode	Int	Blade pitch control mode. 0: No pitch, fix to fine pitch, 1: active PI blade pitch control.
	Y_ControlMode	Int	Yaw control mode. 0: no yaw control, 1: yaw rate control, 2: yaw-by-IPC.
	SS_Mode	Int	Setpoint Smoother mode. 0: no set point smoothing, 1: use set point smoothing.
	WE_Mode	Int	Wind speed estimator mode. 0: One-second low pass filtered hub height wind speed, 1: Immersion and Invariance Estimator, 2: Extended Kalman Filter.
	PS_Mode	Int	Pitch saturation mode. 0: no pitch saturation, 1: implement pitch saturation
	SD_Mode	Int	Shutdown mode. 0: no shutdown procedure, 1: shutdown triggered by max blade pitch.
	Fl_Mode	Int	Floating feedback mode. 0: no nacelle velocity feedback, 1: nacelle velocity feedback (parallel compensation).
	Flp_Mode	Int	Flap control mode. 0: no flap control, 1: steady state flap angle, 2: PI flap control.
FILTERS	F_LPFCornerFreq	Float	Corner frequency (-3dB point) in the generator speed low-pass filter, [rad/s]
	F_LPFDamping	Float	Damping coefficient in the generator speed low-pass filter, [-]. Only used only when F_FilterType = 2
	F_NotchCornerFreq	Float	Natural frequency of the notch filter, [rad/s]
	F_NotchBetaNumDen	Float, Float	Notch damping values of numerator and denominator - determines the width and depth of the notch, [-]
	F_SSCornerFreq	Float	Corner frequency (-3dB point) in the first order low pass ..filter for the set point smoother, [rad/s].
	F_FlCornerFreq	Float, Float	Corner frequency and damping ratio for the second order low pass filter of the tower-top fore-aft motion for floating feedback control [rad/s, -].
	F_FlpCornerFreq	Float, Float	Corner frequency and damping ratio in the second order low pass filter of the blade root bending moment for flap control [rad/s, -].
BLADE PITCH CONTROL	PC_GS_n	Int	Number of gain-scheduling table entries
	PC_GS_angles	Float array, length = PC_GS_n	Gain-schedule table: pitch angles [rad].

continues on next page

Table 4.2 – continued from previous page

Primary Section	Variable	Type	Description
	PC_GS_KP	Float array, length = PC_GS_n	Gain-schedule table: pitch controller proportional gains [s].
	PC_GS_KI	Float array, length = PC_GS_n	Gain-schedule table: pitch controller integral gains [-].
	PC_GS_KD	Float array, length = PC_GS_n	Gain-schedule table: pitch controller derivative gains [s^2]. Currently unused!
	PC_GS_TF	Float array, length = PC_GS_n	Gain-schedule table: transfer function gains [s^2]. Currently unused!
	PC_MaxPit	Float	Maximum physical pitch limit, [rad].
	PC_MinPit	Float	Minimum physical pitch limit, [rad].
	PC_MaxRat	Float	Maximum pitch rate (in absolute value) of pitch controller, [rad/s].
	PC_MinRat	Float	Minimum pitch rate (in absolute value) in pitch controller, [rad/s].
	PC_RefSpd	Float	Desired (reference) HSS speed for pitch controller, [rad/s].
	PC_FinePit	Float	Below-rated pitch angle set-point, [rad]
	PC_Switch	Float	Angle above lowest PC_MinPit to switch to above rated torque control, [rad]. Used for :code:`VS_ControlMode`=0,1.
INDIVIDUAL PITCH CONTROL	IPC_IntSat	Float	Integrator saturation point (maximum signal amplitude contribution to pitch from IPC), [rad]
	IPC_KI	Float Float	Integral gain for the individual pitch controller: first parameter for 1P reductions, second for 2P reductions, [-, -].
	IPC_aziOffset	Float Float	Phase offset added to the azimuth angle for the individual pitch controller: first parameter for 1P reductions, second for 2P reductions, [rad].
	IPC_CornerFreqAct	Float	Corner frequency of the first-order actuators model, used to induce a phase lag in the IPC signal [rad/s]. 0: Disable.
VS TORQUE CONTROL	VS_GenEff	Float	Generator efficiency from mechanical power -> electrical power, [should match the efficiency defined in the generator properties!], [%]
	VS_ArSatTq	Float	Above rated generator torque PI control saturation limit, [Nm].
	VS_MaxRat	Float	Maximum generator torque rate (in absolute value) [Nm/s].
	VS_MaxTq	Float	Maximum generator torque (HSS), [Nm].
	VS_MinTq	Float	Minimum generator torque (HSS) [Nm].
	VS_MinOMSpd	Float	Cut-in speed towards optimal mode gain path, [rad/s]. Used if VS_ControlMode = 0,1.
	VS_Rgn2K	Float	Generator torque constant in Region 2 (HSS side), [N-m/(rad/s)^2]. Used if VS_ControlMode = 0,1.

continues on next page

Table 4.2 – continued from previous page

Primary Section	Variable	Type	Description
	VS_RtPwr	Float	Rated power [W]
	VS_RtTq	Float	Rated torque, [Nm].
	VS_RefSpd	Float	Rated generator speed used by torque controller [rad/s].
	VS_n	Int	Number of generator PI torque controller gains. Only 1 is currently supported.
	VS_KP	Float	Proportional gain for generator PI torque controller [1/(rad/s) Nm]. (Used in the transition 2.5 region if VS_ControlMode = 0,1. Always used if VS_ControlMode = 2,3)
	VS_KI	Float	Integral gain for generator PI torque controller [1/rad Nm]. (Only used in the transition 2.5 region if VS_ControlMode = 0,1. Always used if VS_ControlMode = 2,3)
	VS_TSRopt	Float	Region 2 tip-speed-ratio [rad]. Generally, the power maximizing TSR. Can use non-optimal TSR for low axial induction rotors.
SETPOINT SMOOTHER	SS_VSGain	Float	Variable speed torque controller setpoint smoother gain, [-].
	SS_PCGain	Float	Collective pitch controller setpoint smoother gain, [-].
WIND SPEED ESTIMATOR	WE_BladeRadius	Float	Blade length (distance from hub center to blade tip), [m]
	WE_CP_n	Int	Number of parameters in the Cp array
	WE_CP	Float Float Float Float	Parameters that define the parameterized CP(lambda) function
	WE_Gamma	Float	Adaption gain for the I&I wind speed estimator algorithm [m/rad]
	WE_GearboxRatio	Float	Gearbox ratio [>=1], [-]
	WE_Jtot	Float	Total drivetrain inertia, including blades, hub and casted generator inertia to LSS, [kg m^2]
	WE_RhoAir	Float	Air density, [kg m^-3]
	PerfFileName	String	File containing rotor performance tables (Cp,Ct,Cq)
	PerfTableSize	Int	Size of rotor performance tables in PerfFileName, first number refers to number of blade pitch angles (num columns), second number refers to number of tip-speed ratios (num rows)
	WE_FOPoles_N	Int	Number of first-order system poles used in the Extended Kalman Filter
	WE_FOPoles_v	Float array, length = WE_FOPoles_N	Wind speeds for first-order system poles lookup table [m/s]
	WE_FOPoles	Float array, length = WE_FOPoles_N	First order system poles [1/s]
YAW CONTROL	Y_ErrThresh	Float	Yaw error threshold. Turbine begins to yaw when it passes this. [rad^2 s]

continues on next page

Table 4.2 – continued from previous page

Primary Section	Variable	Type	Description
	Y_IPC_IntSat	Float	Integrator saturation (maximum signal amplitude contribution to pitch from yaw-by-IPC), [rad]
	Y_IPC_n	Int	Number of controller gains for yaw-by-IPC
	Y_IPC_KP	Float array, length = Y_IPC_n	Yaw-by-IPC proportional controller gains Kp [s]
	Y_IPC_KI	Float array, length = Y_IPC_n	Yaw-by-IPC integral controller gain Ki [-]
	Y_IPC_omegaLP	Float	Low-pass filter corner frequency for the Yaw-by-IPC controller to filtering the yaw alignment error, [rad/s].
	Y_IPC_zetaLP	Float	Low-pass filter damping factor for the Yaw-by-IPC controller to filtering the yaw alignment error, [-].
	Y_MErrSet	Float	Yaw alignment error set point, [rad].
	Y_omegaLPFast	Float	Corner frequency fast low pass filter, [rad/s].
	Y_omegaLPSlow	Float	Corner frequency slow low pass filter, [rad/s].
	Y_Rate	Float	Yaw rate, [rad/s].
TOWER FORE-AFT DAMPING	FA_KI	Float	Integral gain for the fore-aft tower damper controller [rad*s/m]. -1 = off
	FA_HPF_CornerFreq	Float	Corner frequency (-3dB point) in the high-pass filter on the fore-aft acceleration signal [rad/s]
	FA_IntSat	Float	Integrator saturation (maximum signal amplitude contribution to pitch from FA damper), [rad]
MINIMUM PITCH SATURATION	PS_BldPitchMin_N	Int	Number of values in minimum blade pitch lookup table.
	PS_WindSpeeds	Float array, length = PS_BldPitchMin_n	Wind speeds corresponding to minimum blade pitch angles [m/s]
	PS_BldPitchMin	Float array, length = PS_BldPitchMin_n	Minimum blade pitch angles [rad]
SHUTDOWN	SD_MaxPit	Float	Maximum blade pitch angle to initiate shutdown, [rad]
	SD_CornerFreq	Float	Cutoff Frequency for first order low-pass filter for blade pitch angle, [rad/s]
FLOATING	Fl_Kp	Float	Nacelle velocity proportional feedback gain [s]

continues on next page

Table 4.2 – continued from previous page

Primary Section	Variable	Type	Description
FLAP ACTUATION	`Flp_Angle`	Float	Initial or steady state flap angle [rad]
	`Flp_Kp`	Float	Trailing edge flap control proportional gain [s]
	`Flp_Ki`	Float	Trailing edge flap control integral gain [s]
	`Flp_MaxPit`	Float	Maximum (and minimum) flap angle [rad]