ROSCO

Release 2.8.0

Nikhar J. Abbas, Daniel S. Zalkind

May 19, 2023

CONTENTS

Version
2.8.0

Date
May 19, 2023

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

Fig. 1 shows the general workflow for the ROSCO tool-chain. with OpenFAST

Fig. 1: ROSCO toolchain general workflow

ROSCO Toolbox The python-based toolbox primarily used for tuning the controller and writing the DISCON.IN.

- Generic tuning of NREL's ROSCO controller
- Simple 1-DOF turbine simulations for quick controller capability verifications
- Parsing of OpenFAST input and output files
- Linear model analysis capability

ROSCO Controller The controller implementation itself. This is compiled to `libdiscon.*` file, reads the DISCON.IN file, and interfaces with OpenFAST using the Bladed-style interface.

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

CHAPTER ONE

STANDARD USE

For the standard use case in OpenFAST (or similar), 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

LICENSE

4.1 Installing the ROSCO tools

As a reminder, the ROSCO toolbox is a python-based tool used to write the DISCON.IN file, which is read by the ROSCO controller (a compiled binary file). If you only wish to run the controller, you *do not* need to install the ROSCO toolbox.

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 tool-chain. Table 4.1 provides an overview of the primary methods available for *Installing the ROSCO controller*. Additionally, Table 4.2 provides an overview of the primary methods available to acquire the ROSCO toolbox. Finally, if you wish to install and use both the controller and toolbox, the section about *Full ROSCO Installation* provides the best methods of doing so.

Table 4.1: Methods for Installing the ROSCO Controller

Method	Use Case
Direct Download	Best for users who simply want to use a released version of the controller without working through the compilation procedures.
Anaconda Download - ROSCO	Best for users who just want to use the controller but prefer to download using the Anaconda package man age Full ROSCO Installation.
Full ROSCO Installation	Best for users who wish to both use the controller and leverage the tools in the ROSCO toolbox
Compile using CMake	Best for users who need to re-compile the source code often, plan to use non-released versions of ROSCO (including modified source code), or who simply want to compile the controller themselves so they have the full code available locally. This is necessary for users who wish to use the *ZeroMQ Interface*.

Table 4.2: Methods for Installing the ROSCO Toolbox

Method	Use Case
Anaconda Download - ROSCO Toolbox	Best for users who simply want to use the primary ROSCO toolbox functions
Full ROSCO Installation	(Recommended) Best for users who wish to both use the primary ROSCO toolbox functions, as well run and use the many example and testing scripts available. This process can be done with or without compiling ROSCO.

For many of the methods used to install both ROSCO and the ROSCO toolbox, both Anaconda and CMake are necessary. Anaconda is a popular package manager used to distribute software packages of various types. Anaconda is used to download requisite packages and distribute pre-compiled versions of the ROSCO tools. CMake is a build configuration system that creates files as input to a build tool like GNU Make, Visual Studio, or Ninja. CMake does not compile code or run compilers directly, but rather creates the environment needed for another tool to run compilers and create binaries. CMake is used to ease the processes of compiling the ROSCO controller locally. For more information on CMake, please see understanding CMake in the OpenFAST documentation.

4.1.1 Installing the ROSCO controller

The standard ROSCO controller is based in Fortran and must be compiled; the source code can be found at: https://github.com/NREL/ROSCO/ROSCO.

Direct Download

The most recent tagged version releases of the controller are available for download. One can simply download these compiled binary files for their system and point to them in their simulation tools (e.g. through `DLL_FileName` in the ServoDyn input file of OpenFAST).

Anaconda Download - ROSCO

Using the popular package manager, Anaconda, the tagged 64-bit versions of ROSCO are 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 <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.* <desired_folder>
```

on linux or:

```
copy %CONDA_PREFIX%/lib/libdiscon.* <desired_folder>
```

on Windows.

Compile using CMake

CMake eases the compiling process significantly. We recommend that users use CMake if at all possible, as we cannot guarantee support for the use of other tools to aid with compiling ROSCO.

On Mac/Linux, standard compilers are generally available without any additional downloads. On 32-bit windows, we recommend that you install MinGW (Section 2). On 64-bit Windows, you can simply install the MSYS2 toolchain through Anaconda:

```
conda install m2w64-toolchain libpython
conda install cmake make  # if Windows users would like to install these in anaconda␣
↪environment
```

Once the CMake and the required compilers are downloaded, the following code can be used to compile ROSCO.

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

# Compile ROSCO
cd ROSCO/ROSCO
mkdir build
cd build
cmake ..                        # Mac/linux only
cmake .. -G "MinGW Makefiles"   # Windows only
make install
```

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

ZeroMQ Interface

There is an option to interface ROSCO with external inputs using ZeroMQ. Currently, only externally commanded yaw offset inputs are supported, though this could easily be expanded if the need arises.

To use the ZeroMQ interface, the software must be downloaded following the ZeroMQ download instructions. Using CMake, ROSCO can then be compiled to enable this interface by using the `ZMQ_CLIENT:ON` flag with the `cmake` command in *Compile using CMake*:

```
cmake -DZMQ_CLIENT:ON ..
```

4.1.2 Installing the ROSCO toolbox

The ROSCO toolbox is based in python and contains all relevant ROSCO tools; the source code can be found at: https://github.com/NREL/ROSCO/. In addition to tuning procedures, the ROSCO toolbox also contains example scripts, a Simulink Model of ROSCO, OpenFAST pre-and post-processing functions, linearized systems analysis tools, and a testing suite.

Anaconda Download - ROSCO Toolbox

If one wishes to simply use the modules provided in the ROSCO toolbox through scripts of their own, the ROSCO toolbox can be installed via the conda-forge channel of Anaconda. They can then be accessed using the standard methods of loading modules in python, e.g:

```
from ROSCO_toolbox import controller as ROSCO_controller
from ROSCO_toolbox import turbine as ROSCO_turbine
```

Note that the install procedures for the ROSCO toolbox are the same as in *Anaconda Download - ROSCO*, but do not involve moving the controller binary file. 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 <desired_folder>
```

and download the controller:

```
conda install -y ROSCO
```

4.1.3 Full ROSCO Installation

We recommend using the full ROSCO tool-chain. This allows for full use of the provided functions along with the developed python packages and controller code,

Please follow the following steps to install the ROSCO tool-chain. You should do step 2 *or* 3. If you simply want to install the ROSCO toolbox without the controller, do step 3. If you would like to install the ROSCO toolbox and compile the controller simultaneously, do step 2.

1. Create a conda environment for ROSCO

```
conda config --add channels conda-forge # (Enable Conda-forge Channel For Conda Package␣
↪Manager)
conda create -y --name rosco-env python=3.8 # (Create a new environment named "rosco-env
↪" that contains Python 3.8)
conda activate rosco-env # (Activate your "rosco-env" environment)
```

2. Clone and Install the ROSCO toolbox with ROSCO controller

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

(continues on next page)

(continued from previous page)

```
conda install compilers # (Mac/Linux only)
conda install m2w64-toolchain libpython # (Windows only)
conda env config vars set FC=gfortran # Sometimes needed for Windows
conda install -y wisdem=3.5.0
python setup.py install --compile-rosco
```

3. Clone and Install the ROSCO toolbox without ROSCO controller

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

4.1.4 Getting Started

Please see *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 the ROSCO toolbox and 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 *ROSCO_Toolbox tuning .yaml*, respectively.

- `01_turbine_model.py` loads an OpenFAST turbine model and displays a summary of its information

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

- `02_ccblade.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.

- `03_tune_controller.py` loads a turbine and tunes the PI control gains
- `04_simple_sim.py` tunes a controller and runs a simple simualtion (not using OpenFAST)
- `05_openfast_sim.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:

- `06_peak_shaving.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):

- `09_distributed_aero.py` tunes a controller for distributed aerodynamic control

The ROSCO toolbox also contains methods for working with OpenFAST linear models * `10_linear_params.py` exports a file of the parameters used for the simplified linear models used to tune ROSCO * `11_robust_tuning.py` shows how linear models generated using OpenFAST can be used to tune controllers with robust stability properties. * `12_tune_ipc.py` shows the tuning procedure for IPC

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:

- `05_openfast_sim.py` loads a turbine, tunes a controller, and runs an OpenFAST simulation
- `07_openfast_outputs.py` loads the OpenFAST output files and plots the results
- `08_run_turbsim.py` runs TurbSim, for generating turbulent wind inputs
- `14_open_loop_control.py` runs an OpenFAST simulation with ROSCO providing open loop control 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 ROSCO Toolbox 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 can be found here: *ROSCO_Toolbox tuning .yaml*.

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 ROSCO 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 4.3: DISCON.IN

Primary Section	Variable	Type	Description
DEBUG	`LoggingL`	Int	0: write no debug files, 1: write standard output .dbg-file, 2: write standard output .dbg-file and complete avrSWAP-array .dbg2-file
CONTROLLER FLAGS	`F_LPFTyp`	Int	Filter type for generator speed feedback signal. 1: first-order low-pass filter, 2: second-order low-pass filter.
	`F_NotchT`	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_Cont`	Int	Individual Pitch Control (IPC) type for fatigue load reductions (pitch contribution). 0: off, 1: 1P reductions, 2: 1P+2P reductions.
	`VS_Contr`	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_Contr`	Int	Blade pitch control mode. 0: No pitch, fix to fine pitch, 1: active PI blade pitch control.
	`Y_Contro`	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_LPFCor`	Float	Corner frequency (-3dB point) in the generator speed low-pass filter, [rad/s]
	`F_LPFDam`	Float	Damping coefficient in the generator speed low-pass filter, [-]. Only used only when F_FilterType = 2
	`F_NotchC`	Float	Natural frequency of the notch filter, [rad/s]
	`F_NotchB`	Float Float	Notch damping values of numerator and denominator - determines the width and depth of the notch, [-]

continues on next page

Table 4.3 – continued from previous page

Primary Section	Variable	Type	Description
	F_SSCorn	Float	Corner frequency (-3dB point) in the first order low pass ..filter for the set point smoother, [rad/s].
	F_FlCorn	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_WECorn	Float	Corner frequency (-3dB point) in the first order low pass filter for the wind speed estimate [rad/s].
	F_FlHigh	Float	Natural frequency of first-order high-pass filter for nacelle fore-aft motion [rad/s]..
	F_FlpCor	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_an	Float array, length = PC_GS_n	Gain-schedule table: pitch angles [rad].
	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_MaxPi	Float	Maximum physical pitch limit, [rad].
	PC_MinPi	Float	Minimum physical pitch limit, [rad].
	PC_MaxRa	Float	Maximum pitch rate (in absolute value) of pitch controller, [rad/s].
	PC_MinRa	Float	Minimum pitch rate (in absolute value) in pitch controller, [rad/s].
	PC_RefSp	Float	Desired (reference) HSS speed for pitch controller, [rad/s].
	PC_FineP	Float	Below-rated pitch angle set-point, [rad]
	PC_Switc	Float	Angle above lowest `PC_MinPit` to switch to above rated torque control, [rad]. Used for `VS_ControlMode` = 0,1.
INDIVIDUAL PITCH CONTROL	IPC_IntS	Float	Integrator saturation point (maximum signal amplitude contribution to pitch from IPC), [rad]

continues on next page

Table 4.3 – continued from previous page

Primary Section	Variable	Type	Description
	IPC_KI	Float Float	Integral gain for the individual pitch controller: first parameter for 1P reductions, second for 2P reductions, [-, -].
	IPC_aziC	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_Corn	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_GenEf	Float	Generator efficiency from mechanical power -> electrical power, [should match the efficiency defined in the generator properties!], [%]
	VS_ArSat	Float	Above rated generator torque PI control saturation limit, [Nm].
	VS_MaxRa	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_MinOM	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.
	VS_RtPwr	Float	Rated power [W]
	VS_RtTq	Float	Rated torque, [Nm].
	VS_RefSp	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_TSRop	Float	Region 2 tip-speed-ratio [rad]. Generally, the power maximizing TSR. Can use non-optimal TSR for low axial induction rotors.
SETPOINT SMOOTH	SS_VSGai	Float	Variable speed torque controller setpoint smoother gain, [-].
	SS_PCGai	Float	Collective pitch controller setpoint smoother gain, [-].
WIND SPEED ESTIMATOR	WE_Blade	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_Gearb	Float	Gearbox ratio [>=1], [-]
	WE_Jtot	Float	Total drivetrain inertia, including blades, hub and casted generator inertia to LSS, [kg m^2]
	WE_RhoAi	Float	Air density, [kg m^-3]
	PerfFile	String	File containing rotor performance tables (Cp,Ct,Cq)

continues on next page

Table 4.3 – continued from previous page

Primary Section	Variable	Type	Description
	PerfTabl	Int 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_FOPol	Int	Number of first-order system poles used in the Extended Kalman Filter
	WE_FOPol	Float array, length = WE_FOPol	Wind speeds for first-order system poles lookup table [m/s]
	WE_FOPol	Float array, length = WE_FOPol	First order system poles [1/s]
YAW CONTROL	Y_ErrThr	Float	Yaw error threshold. Turbine begins to yaw when it passes this. [rad^2 s]
	Y_IPC_In	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_om	Float	Low-pass filter corner frequency for the Yaw-by-IPC controller to filtering the yaw alignment error, [rad/s].
	Y_IPC_ze	Float	Low-pass filter damping factor for the Yaw-by-IPC controller to filtering the yaw alignment error, [-].
	Y_MErrSe	Float	Yaw alignment error set point, [rad].
	Y_omegaL	Float	Corner frequency fast low pass filter, [rad/s].
	Y_omegaL	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_C	Float	Corner frequency (-3dB point) in the high-pass filter on the fore-aft acceleration signal [rad/s]
	FA_IntSa	Float	Integrator saturation (maximum signal amplitude contribution to pitch from FA damper), [rad]
MINIMUM PITCH SATURATION	PS_BldPi	Int	Number of values in minimum blade pitch lookup table.

continues on next page

Table 4.3 – continued from previous page

Primary Section	Variable	Type	Description
	`PS_WindS`	Float array, length = `PS_BldPi`	Wind speeds corresponding to minimum blade pitch angles [m/s]
	`PS_BldPi`	Float array, length = `PS_BldPi`	Minimum blade pitch angles [rad]
SHUTDOWN	`SD_MaxPi`	Float	Maximum blade pitch angle to initiate shutdown, [rad]
	`SD_Corne`	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]
FLAP ACTUATION	`Flp_Angl`	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_MaxP`	Float	Maximum (and minimum) flap angle [rad]

4.5 API changes between versions

This page lists the main changes in the ROSCO API (input file) between different versions.

The changes are tabulated according to the line number, and flag name. The line number corresponds to the resulting line number after all changes are implemented. Thus, be sure to implement each in order so that subsequent line numbers are correct.

4.5.1 2.7.0 to 2.8.0

Optional Inputs - ROSCO now reads in the whole input file and searches for keywords to set the inputs. Blank spaces and specific ordering are no longer required. - Input requirements depend on control modes. E.g., open loop inputs are not required if *OL_Mode = 0* ` Cable Control - Can control OpenFAST cables (MoorDyn or SubDyn) using ROSCO Structural Control - Can control OpenFAST structural control elements (ServoDyn) using ROSCO Active wake control - Added Active Wake Control (AWC) implementation

New in ROSCO 2.8.0		
Line	Input Name	Example Value
6	Echo	0 ! Echo - (0 - no Echo, 1 - Echo input data to <RootName>.echo)
25	AWC_Mode	0 ! AWC_Mode - Active wake control mode [0 - not used, 1 - complex number method, 2 - Coleman transform method]
28	CC_Mode	0 ! CC_Mode - Cable control mode [0- unused, 1- User defined, 2- Open loop control]
29	StC_Mode	0 ! StC_Mode - Structural control mode [0- unused, 1- User defined, 2- Open loop control]
139	Ind_CableC	0 ! Ind_CableControl - The column(s) in OL_Filename that contains the cable control inputs in m [Used with CC_Mode = 2, must be the same size as CC_Group_N]
140	Ind_StructC	0 ! Ind_StructControl - The column(s) in OL_Filename that contains the structural control inputs [Used with StC_Mode = 2, must be the same size as StC_Group_N]
148	Empty Line	
149	AWC_Secti	!——- Active Wake Control ————————————————
150	AWC_Num	1 ! AWC_NumModes - AWC- Number of modes to include [-]
151	AWC_n	1 ! AWC_n - AWC azimuthal mode [-] (only used in complex number method)
152	AWC_harm	1 ! AWC_harmonic - AWC Coleman transform harmonic [-] (only used in Coleman transform method)
153	AWC_freq	0.03 ! AWC_freq - AWC frequency [Hz]
154	AWC_amp	2.0 ! AWC_amp - AWC amplitude [deg]
155	AWC_clock	0.0 ! AWC_clockangle - AWC clock angle [deg]
165	Empty Line	
166	CC_Section	!——- Cable Control —————————————————
167	CC_Group_	3 ! CC_Group_N - Number of cable control groups
168	CC_GroupI	2601 2603 2605 ! CC_GroupIndex - First index for cable control group, should correspond to deltaL
169	CC_ActTau	20.000000 ! CC_ActTau - Time constant for line actuator [s]
170	Empty Line	
171	StC_Sectior	!——- Structural Controllers —————————————————
172	StC_Group_	3 ! StC_Group_N - Number of cable control groups
173	StC_GroupI	2818 2838 2858 ! StC_GroupIndex - First index for structural control group, options specified in ServoDyn summary output

4.5.2 2.6.0 to 2.7.0

Pitch Faults - Constant pitch actuator offsets (PF_Mode = 1) IPC Saturation Modes - Added options for saturating the IPC command with the peak shaving limit

New in ROSCO 2.7.0		
Line	Input Name	Example Value
23	PF_Moc	0 ! PF_Mode - Pitch fault mode {0 - not used, 1 - constant offset on one or more blades}
56	IPC_Sat	2 ! IPC_SatMode - IPC Saturation method (0 - no saturation (except by PC_MinPit), 1 - saturate by PS_BldPitchMin, 2 - saturate sotfly (full IPC cycle) by PC_MinPit, 3 - saturate softly by PS_BldPitchMin)
139	PF_Sect	!——- Pitch Actuator Faults ————————————————————
140	PF_Offs	0.00000000 0.00000000 0.00000000 ! PF_Offsets - Constant blade pitch offsets for blades 1-3 [rad]
141	Empty Line	

4.5.3 2.5.0 to 2.6.0

IPC - A wind speed based soft cut-in using a sigma interpolation is added for the IPC controller

Pitch Actuator - A first or second order filter can be used to model a pitch actuator

External Control Interface - Call another control library from ROSCO

ZeroMQ Interface - Communicate with an external routine via ZeroMQ. Only yaw control currently supported

Updated yaw control - Filter wind direction with deadband, and yaw until direction error changes signs (https://iopscience.iop.org/article/10.1088/1742-6596/1037/3/032011)

New in ROSCO 2.6.0		
Line	Input Name	Example Value
19	TD_Mode	0 ! TD_Mode - Tower damper mode {0: no tower damper, 1: feed back translational nacelle accelleration to pitch angle}
22	PA_Mode	0 ! PA_Mode - Pitch actuator mode {0 - not used, 1 - first order filter, 2 - second order filter}
23	Ext_Mode	0 ! Ext_Mode - External control mode {0 - not used, 1 - call external dynamic library}
24	ZMQ_Mode	0 ! ZMQ_Mode - Fuse ZeroMQ interaface {0: unused, 1: Yaw Control}
33	F_YawErr	0.17952 ! F_YawErr - Low pass filter corner frequency for yaw controller [rad/s].
54	IPC_Vramp	9.120000 11.400000 ! IPC_Vramp - Start and end wind speeds for cut-in ramp function. First entry: IPC inactive, second entry: IPC fully active. [m/s]
96	Y_uSwitch	0.00000 ! Y_uSwitch - Wind speed to switch between Y_ErrThresh. If zero, only the first value of Y_ErrThresh is used [m/s]
133	Empty Line	N/A
134	PitchActSec	!——- Pitch Actuator Model ————————————————————
135	PA_CornerFre	3.140000000000 ! PA_CornerFreq - Pitch actuator bandwidth/cut-off frequency [rad/s]
136	PA_Damping	0.707000000000 ! PA_Damping - Pitch actuator damping ratio [-, unused if PA_Mode = 1]
137	Empty Line	
138	ExtConSec	!——- External Controller Interface ————————————————————
139	DLL_FileNam	"unused" ! DLL_FileName - Name/location of the dynamic library in the Bladed-DLL format
140	DLL_InFile	"unused" ! DLL_InFile - Name of input file sent to the DLL (-)
141	DLL_ProcNan	"DISCON" ! DLL_ProcName - Name of procedure in DLL to be called (-)
142	Empty Line	
143	ZeroMQSec	!——- ZeroMQ Interface ————————————————————
144	ZMQ_CommA	"tcp://localhost:5555" ! ZMQ_CommAddress - Communication address for ZMQ server, (e.g. "tcp://localhost:5555")
145	ZMQ_UpdateI	2 ! ZMQ_UpdatePeriod - Call ZeroMQ every [x] seconds, [s]

Modified in ROSCO 2.6.0		
Line	Input Name	Example Value
97	Y_ErrTh	4.000000 8.000000 ! Y_ErrThresh - Yaw error threshold/deadbands. Turbine begins to yaw when it passes this. If Y_uSwitch is zero, only the second value is used. [deg].
98	Y_Rate	0.00870 ! Y_Rate - Yaw rate [rad/s]
99	Y_MErr!	0.00000 ! Y_MErrSet - Integrator saturation (maximum signal amplitude contribution to pitch from yaw-by-IPC), [rad]

Removed in ROSCO 2.6.0		
Line	Input Name	Example Value
96	Y_IPn	1 ! Y_IPC_n - Number of controller gains (yaw-by-IPC)
99	Y_IPC_omeg	0.20940 ! Y_IPC_omegaLP - Low-pass filter corner frequency for the Yaw-by-IPC controller to filtering the yaw alignment error, [rad/s].
100	Y_IPC_zetaI	1.00000 ! Y_IPC_zetaLP - Low-pass filter damping factor for the Yaw-by-IPC controller to filtering the yaw alignment error, [-].
102	Y_omegaLPI	0.20940 ! Y_omegaLPFast - Corner frequency fast low pass filter, 1.0 [rad/s]
103	Y_omegaLP!	0.10470 ! Y_omegaLPSlow - Corner frequency slow low pass filter, 1/60 [rad/s]

4.5.4 ROSCO v2.4.1 to ROSCO v2.5.0

Two filter parameters were added to - change the high pass filter in the floating feedback module - change the low pass filter of the wind speed estimator signal that is used in torque control

Open loop control inputs, users must specify: - The open loop input filename, an example can be found in Examples/Example_OL_Input.dat - Indices (columns) of values specified in OL_Filename

IPC - Proportional Control capabilities were added, 1P and 2P gains should be specified

Line	Flag Name	Example Value
20	OL_Mode	0 ! OL_Mode - Open loop control mode {0: no open loop control, 1: open loop control vs. time, 2: open loop control vs. wind speed}
27	F_WECornei	0.20944 ! F_WECornerFreq - Corner frequency (-3dB point) in the first order low pass filter for the wind speed estimate [rad/s].
29	F_FlHighPas	0.01000 ! F_FlHighPassFreq - Natural frequency of first-order high-pass filter for nacelle fore-aft motion [rad/s].
50	IPC_KP	0.000000 0.000000 ! IPC_KP - Proportional gain for the individual pitch controller: first parameter for 1P reductions, second for 2P reductions, [-]
125	OL_Filename	"14_OL_Input.dat" ! OL_Filename - Input file with open loop timeseries (absolute path or relative to this file)
126	Ind_Breakpo	1 ! Ind_Breakpoint - The column in OL_Filename that contains the breakpoint (time if OL_Mode = 1)
127	Ind_BldPitch	2 ! Ind_BldPitch - The column in OL_Filename that contains the blade pitch input in rad
128	Ind_GenTq	3 ! Ind_GenTq - The column in OL_Filename that contains the generator torque in Nm
129	Ind_YawRate	4 ! Ind_YawRate - The column in OL_Filename that contains the generator torque in Nm

4.6 ROSCO_Toolbox tuning .yaml

Definition of inputs for ROSCO tuning procedure

toolbox_schema

4.6.1 path_params

FAST_InputFile
    [String] Name of **.fst* file

FAST_directory
    [String] Main OpenFAST model directory, where the **.fst* lives, relative to ROSCO dir (if applicable)

rotor_performance_filename
    [String] Filename for rotor performance text file (if it has been generated by ccblade already)

4.6.2 turbine_params

rotor_inertia
    [Float, kg m^2] Rotor inertia [kg m^2], {Available in Elastodyn .sum file}

rated_rotor_speed
    [Float, rad/s] Rated rotor speed [rad/s]

    Minimum = 0

v_min
    [Float, m/s] Cut-in wind speed of the wind turbine.

    Minimum = 0

v_max
    [Float, m/s] Cut-out wind speed of the wind turbine.

    Minimum = 0

max_pitch_rate
    [Float, rad/s] Maximum blade pitch rate [rad/s]

    Minimum = 0

max_torque_rate
    [Float, Nm/s] Maximum torque rate [Nm/s], {~1/4 VS_RtTq/s}

    Minimum = 0

rated_power
    [Float, W] Rated Power [W]

    Minimum = 0

bld_edgewise_freq
    [Float, rad/s] Blade edgewise first natural frequency [rad/s]

    Default = 4.0

    Minimum = 0

`bld_flapwise_freq`
[Float, rad/s] Blade flapwise first natural frequency [rad/s]

Default = 0

Minimum = 0

`TSR_operational`
[Float] Optimal tip speed ratio, if 0 the optimal TSR will be determined by the Cp surface

Default = 0

Minimum = 0

4.6.3 controller_params

`LoggingLevel`
[Float] 0- write no debug files, 1- write standard output .dbg-file, 2- write standard output .dbg-file and complete avrSWAP-array .dbg2-file

Default = 1

Minimum = 0 *Maximum* = 3

`F_LPFType`
[Float] 1- first-order low-pass filter, 2- second-order low-pass filter, [rad/s] (currently filters generator speed and pitch control signals)

Default = 1

Minimum = 1 *Maximum* = 2

`F_NotchType`
[Float] Notch on the measured generator speed and/or tower fore-aft motion (for floating) {0- disable, 1- generator speed, 2- tower-top fore- aft motion, 3- generator speed and tower-top fore-aft motion}

Default = 0

Minimum = 0 *Maximum* = 3

`IPC_ControlMode`
[Float] Turn Individual Pitch Control (IPC) for fatigue load reductions (pitch contribution) (0- off, 1- 1P reductions, 2- 1P+2P reduction)

Default = 0

Minimum = 0 *Maximum* = 2

`VS_ControlMode`
[Float] Generator torque control mode in above rated conditions (0- constant torque, 1- constant power, 2- TSR tracking PI control with constant torque, 3- TSR tracking with constant power)

Default = 2

Minimum = 0 *Maximum* = 3

`PC_ControlMode`
[Float] Blade pitch control mode (0- No pitch, fix to fine pitch, 1- active PI blade pitch control)

Default = 1

Minimum = 0 *Maximum* = 1

`Y_ControlMode`
: [Float] Yaw control mode (0- no yaw control, 1- yaw rate control, 2- yaw- by-IPC)

  Default = 0

  Minimum = 0 *Maximum* = 2

`SS_Mode`
: [Float] Setpoint Smoother mode (0- no setpoint smoothing, 1- introduce setpoint smoothing)

  Default = 1

  Minimum = 0 *Maximum* = 2

`WE_Mode`
: [Float] Wind speed estimator mode (0- One-second low pass filtered hub height wind speed, 1- Immersion and Invariance Estimator (Ortega et al.)

  Default = 2

  Minimum = 0 *Maximum* = 2

`PS_Mode`
: [Float] Pitch saturation mode (0- no pitch saturation, 1- peak shaving, 2- Cp-maximizing pitch saturation, 3- peak shaving and Cp-maximizing pitch saturation)

  Default = 3

  Minimum = 0 *Maximum* = 3

`SD_Mode`
: [Float] Shutdown mode (0- no shutdown procedure, 1- pitch to max pitch at shutdown)

  Default = 0

  Minimum = 0 *Maximum* = 1

`TD_Mode`
: [Float] Tower damper mode (0- no tower damper, 1- feed back translational nacelle accelleration to pitch angle

  Default = 0

  Minimum = 0 *Maximum* = 1

`Fl_Mode`
: [Float] Floating specific feedback mode (0- no nacelle velocity feedback, 1 - nacelle velocity feedback, 2 - nacelle pitching acceleration feedback)

  Default = 0

  Minimum = 0 *Maximum* = 2

`Flp_Mode`
: [Float] Flap control mode (0- no flap control, 1- steady state flap angle, 2- Proportional flap control)

  Default = 0

  Minimum = 0 *Maximum* = 2

`PwC_Mode`
: [Float] Active Power Control Mode (0- no active power control 1- constant active power control, 2- open loop power vs time, 3- open loop power vs. wind speed)

  Default = 0

  Minimum = 0 *Maximum* = 2

ZMQ_Mode
[Float] ZMQ Mode (0 - ZMQ Inteface, 1 - ZMQ for yaw control)

Default = 0

Minimum = 0 *Maximum* = 1

PA_Mode
[Float] Pitch actuator mode {0 - not used, 1 - first order filter, 2 - second order filter}

Default = 0

Minimum = 0 *Maximum* = 2

PF_Mode
[Float] Pitch fault mode {0 - not used, 1 - constant offset on one or more blades}

Default = 0

Minimum = 0 *Maximum* = 1

Ext_Mode
[Float] External control mode [0 - not used, 1 - call external dynamic library]

Default = 0

Minimum = 0 *Maximum* = 1

CC_Mode
[Float] Cable control mode [0- unused, 1- User defined, 2- Position control (not yet implemented)]

Default = 0

Minimum = 0 *Maximum* = 1

StC_Mode
[Float] Structural control mode [0- unused, 1- User defined]

Default = 0

Minimum = 0 *Maximum* = 1

U_pc
[Array of Floats] List of wind speeds to schedule pitch control zeta and omega

Default = [12]

Minimum = 0

zeta_pc
[Array of Floats or Float] List of pitch controller desired damping ratio at U_pc [-]

Default = [1.0]

omega_pc
[Array of Floats or Float, rad/s] List of pitch controller desired natural frequency at U_pc [rad/s]

Default = [0.2]

interp_type
[String from, ['sigma', 'linear', 'quadratic', 'cubic']] Type of interpolation between above rated tuning values (only used for multiple pitch controller tuning values)

Default = sigma

zeta_vs
[Float] Torque controller desired damping ratio [-]

Default = 1.0

Minimum = 0

omega_vs
[Float, rad/s] Torque controller desired natural frequency [rad/s]

Default = 0.2

Minimum = 0

max_pitch
[Float, rad] Maximum pitch angle [rad], {default = 90 degrees}

Default = 1.57

min_pitch
[Float, rad] Minimum pitch angle [rad], {default = 0 degrees}

Default = 0

vs_minspd
[Float, rad/s] Minimum rotor speed [rad/s], {default = 0 rad/s}

Default = 0

ss_vsgain
[Float] Torque controller setpoint smoother gain bias percentage [%, <= 1], {default = 100%}

Default = 1.0

ss_pcgain
[Float, rad] Pitch controller setpoint smoother gain bias percentage [%, <= 1], {default = 0.1%}

Default = 0.001

ps_percent
[Float, rad] Percent peak shaving [%, <= 1], {default = 80%}

Default = 0.8 *Maximum* = 1

sd_maxpit
[Float, rad] Maximum blade pitch angle to initiate shutdown [rad], {default = 40 deg.}

Default = 0.6981

flp_maxpit
[Float, rad] Maximum (and minimum) flap pitch angle [rad]

Default = 0.1745

twr_freq
[Float, rad/s] Tower natural frequency, for floating only

Minimum = 0

ptfm_freq
[Float, rad/s] Platform natural frequency, for floating only

Minimum = 0

WS_GS_n
[Float] Number of wind speed breakpoints

Default = 60

Minimum = 0

PC_GS_n
[Float] Number of pitch angle gain scheduling breakpoints

Default = 30

Minimum = 0

Kp_float
[Float, s] Gain of floating feedback control

tune_Fl
[Boolean] Whether to automatically tune Kp_float

Default = True

zeta_flp
[Float] Flap controller desired damping ratio [-]

Minimum = 0

omega_flp
[Float, rad/s] Flap controller desired natural frequency [rad/s]

Minimum = 0

flp_kp_norm
[Float] Flap controller normalization term for DC gain (kappa)

Minimum = 0

flp_tau
[Float, s] Flap controller time constant for integral gain

Minimum = 0

max_torque_factor
[Float] Maximum torque = rated torque * max_torque_factor

Default = 1.1

Minimum = 0

IPC_Kp1p
[Float, s] Proportional gain for IPC, 1P [s]

Default = 0.0

Minimum = 0

IPC_Kp2p
[Float] Proportional gain for IPC, 2P [-]

Default = 0.0

Minimum = 0

IPC_Ki1p
[Float, s] Integral gain for IPC, 1P [s]

Default = 0.0

Minimum = 0

IPC_Ki2p
[Float] integral gain for IPC, 2P [-]

Default = 0.0

Minimum = 0

IPC_Vramp
[Array of Floats] wind speeds for IPC cut-in sigma function [m/s]

Default = [0.0, 0.0]

Minimum = 0.0

filter_params

f_lpf_cornerfreq
[Float, rad/s] Corner frequency (-3dB point) in the first order low pass filter of the generator speed [rad/s]

Minimum = 0

f_lpf_damping
[Float, rad/s] Damping ratio in the first order low pass filter of the generator speed [-]

Minimum = 0

f_we_cornerfreq
[Float, rad/s] Corner frequency (-3dB point) in the first order low pass filter for the wind speed estimate [rad/s]

Default = 0.20944

Minimum = 0

f_fl_highpassfreq
[Float, rad/s] Natural frequency of first-order high-pass filter for nacelle fore-aft motion [rad/s]

Default = 0.01042

Minimum = 0

f_ss_cornerfreq
[Float, rad/s] First order low-pass filter cornering frequency for setpoint smoother [rad/s]

Default = 0.6283

Minimum = 0

f_yawerr
[Float, rad/s] Low pass filter corner frequency for yaw controller [rad/

Default = 0.17952

Minimum = 0

f_sd_cornerfreq
[Float, rad] Cutoff Frequency for first order low-pass filter for blade pitch angle [rad/s], {default = 0.41888 ~ time constant of 15s}

Default = 0.41888

open_loop

`flag`
: [Boolean] Flag to use open loop control

  Default = False

`filename`
: [String] Filename of open loop input that ROSCO reads

  Default = unused

`OL_Ind_Breakpoint`
: [Float] Index (column, 1-indexed) of breakpoint (time) in open loop index

  Default = 1

`OL_Ind_BldPitch`
: [Float] Index (column, 1-indexed) of breakpoint (time) in open loop index

  Default = 0

`OL_Ind_GenTq`
: [Float] Index (column, 1-indexed) of breakpoint (time) in open loop index

  Default = 0

`OL_Ind_YawRate`
: [Float] Index (column, 1-indexed) of breakpoint (time) in open loop index

  Default = 0

`PA_CornerFreq`
: [Float, rad/s] Pitch actuator natural frequency [rad/s]

  Default = 3.14

  Minimum = 0

`PA_Damping`
: [Float] Pitch actuator damping ratio [-]

  Default = 0.707

  Minimum = 0

DISCON

These are pass-through parameters for the DISCON.IN file. Use with caution.

`LoggingLevel`
: [Float] (0- write no debug files, 1- write standard output .dbg-file, 2- write standard output .dbg-file and complete avrSWAP-array .dbg2-file)

`Echo`
: [Float] 0 - no Echo, 1 - Echo input data to <RootName>.echo

  Default = 0

`F_LPFType`
: [Float] 1- first-order low-pass filter, 2- second-order low-pass filter (currently filters generator speed and pitch control signals

F_NotchType
[Float] Notch on the measured generator speed and/or tower fore-aft motion (for floating) (0- disable, 1- generator speed, 2- tower-top fore- aft motion, 3- generator speed and tower-top fore-aft motion)

IPC_ControlMode
[Float] Turn Individual Pitch Control (IPC) for fatigue load reductions (pitch contribution) (0- off, 1- 1P reductions, 2- 1P+2P reductions)

VS_ControlMode
[Float] Generator torque control mode in above rated conditions (0- constant torque, 1- constant power, 2- TSR tracking PI control with constant torque, 3- TSR tracking PI control with constant power)

PC_ControlMode
[Float] Blade pitch control mode (0- No pitch, fix to fine pitch, 1- active PI blade pitch control)

Y_ControlMode
[Float] Yaw control mode (0- no yaw control, 1- yaw rate control, 2- yaw- by-IPC)

SS_Mode
[Float] Setpoint Smoother mode (0- no setpoint smoothing, 1- introduce setpoint smoothing)

WE_Mode
[Float] 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
[Float] Pitch saturation mode (0- no pitch saturation, 1- implement pitch saturation)

SD_Mode
[Float] Shutdown mode (0- no shutdown procedure, 1- pitch to max pitch at shutdown)

Fl_Mode
[Float] Floating specific feedback mode (0- no nacelle velocity feedback, 1- feed back translational velocity, 2- feed back rotational veloicty)

Flp_Mode
[Float] Flap control mode (0- no flap control, 1- steady state flap angle, 2- Proportional flap control)

F_LPFCornerFreq
[Float, rad/s] Corner frequency (-3dB point) in the low-pass filters,

F_LPFDamping
[Float] Damping coefficient (used only when F_FilterType = 2 [-]

F_NotchCornerFreq
[Float, rad/s] Natural frequency of the notch filter,

F_NotchBetaNumDen
[Array of Floats] Two notch damping values (numerator and denominator, resp) - determines the width and depth of the notch, [-]

F_SSCornerFreq
[Float, rad/s.] Corner frequency (-3dB point) in the first order low pass filter for the setpoint smoother,

F_WECornerFreq
[Float, rad/s.] Corner frequency (-3dB point) in the first order low pass filter for the wind speed estimate

F_FlCornerFreq
[Array of Floats] Natural frequency and damping in the second order low pass filter of the tower-top fore-aft motion for floating feedback control

F_FlHighPassFreq
[Float, rad/s] Natural frequency of first-order high-pass filter for nacelle fore-aft motion

F_FlpCornerFreq
[Array of Floats] Corner frequency and damping in the second order low pass filter of the blade root bending moment for flap control

PC_GS_n
[Float] Amount of gain-scheduling table entries

PC_GS_angles
[Array of Floats] Gain-schedule table- pitch angles

PC_GS_KP
[Array of Floats] Gain-schedule table- pitch controller kp gains

PC_GS_KI
[Array of Floats] Gain-schedule table- pitch controller ki gains

PC_GS_KD
[Array of Floats] Gain-schedule table- pitch controller kd gains

PC_GS_TF
[Array of Floats] Gain-schedule table- pitch controller tf gains (derivative filter)

PC_MaxPit
[Float, rad] Maximum physical pitch limit,

PC_MinPit
[Float, rad] Minimum physical pitch limit,

PC_MaxRat
[Float, rad/s.] Maximum pitch rate (in absolute value) in pitch controller

PC_MinRat
[Float, rad/s.] Minimum pitch rate (in absolute value) in pitch controller

PC_RefSpd
[Float, rad/s.] Desired (reference) HSS speed for pitch controller

PC_FinePit
[Float, rad] Record 5- Below-rated pitch angle set-point

PC_Switch
[Float, rad] Angle above lowest minimum pitch angle for switch

IPC_IntSat
[Float, rad] Integrator saturation (maximum signal amplitude contribution to pitch from IPC)

IPC_SatMode
[Integer] IPC Saturation method (0 - no saturation, 1 - saturate by PC_MinPit, 2 - saturate by PS_BldPitchMin)

IPC_KP
[Array of Floats] Proportional gain for the individual pitch controller- first parameter for 1P reductions, second for 2P reductions, [-]

IPC_KI
[Array of Floats] Integral gain for the individual pitch controller- first parameter for 1P reductions, second for 2P reductions, [-]

IPC_aziOffset
[Array of Floats] Phase offset added to the azimuth angle for the individual pitch controller

IPC_CornerFreqAct
[Float, rad/s] Corner frequency of the first-order actuators model, to induce a phase lag in the IPC signal (0-Disable)

VS_GenEff
[Float, percent] Generator efficiency mechanical power -> electrical power, should match the efficiency defined in the generator properties

VS_ArSatTq
[Float, Nm] Above rated generator torque PI control saturation

VS_MaxRat
[Float, Nm/s] Maximum torque rate (in absolute value) in torque controller

VS_MaxTq
[Float, Nm] Maximum generator torque in Region 3 (HSS side)

VS_MinTq
[Float, Nm] Minimum generator torque (HSS side)

VS_MinOMSpd
[Float, rad/s] Minimum generator speed

VS_Rgn2K
[Float, Nm/(rad/s)^2] Generator torque constant in Region 2 (HSS side)

VS_RtPwr
[Float, W] Wind turbine rated power

VS_RtTq
[Float, Nm] Rated torque

VS_RefSpd
[Float, rad/s] Rated generator speed

VS_n
[Float] Number of generator PI torque controller gains

VS_KP
[Float] Proportional gain for generator PI torque controller. (Only used in the transitional 2.5 region if VS_ControlMode =/ 2)

VS_KI
[Float, s] Integral gain for generator PI torque controller (Only used in the transitional 2.5 region if VS_ControlMode =/ 2)

VS_TSRopt
[Float, rad] Power-maximizing region 2 tip-speed-ratio

SS_VSGain
[Float] Variable speed torque controller setpoint smoother gain

SS_PCGain
[Float] Collective pitch controller setpoint smoother gain

WE_BladeRadius
[Float, m] Blade length (distance from hub center to blade tip)

WE_CP_n
[Float] Amount of parameters in the Cp array

WE_CP
[Array of Floats] Parameters that define the parameterized CP(lambda) function

WE_Gamma
[Float, m/rad] Adaption gain of the wind speed estimator algorithm

WE_GearboxRatio
[Float] Gearbox ratio, >=1

WE_Jtot
[Float, kg m^2] Total drivetrain inertia, including blades, hub and casted generator inertia to LSS

WE_RhoAir
[Float, kg m^-3] Air density

PerfFileName
[String] File containing rotor performance tables (Cp,Ct,Cq) (absolute path or relative to this file)

PerfTableSize
[Float] Size of rotor performance tables, first number refers to number of blade pitch angles, second number referse to number of tip-speed ratios

WE_FOPoles_N
[Float] Number of first-order system poles used in EKF

WE_FOPoles_v
[Array of Floats] Wind speeds corresponding to first-order system poles

WE_FOPoles
[Array of Floats] First order system poles

Y_ErrThresh
[Float, rad^2 s] Yaw error threshold. Turbine begins to yaw when it passes this

Y_IPC_IntSat
[Float, rad] Integrator saturation (maximum signal amplitude contribution to pitch from yaw-by-IPC)

Y_IPC_n
[Float] Number of controller gains (yaw-by-IPC)

Y_IPC_KP
[Float] Yaw-by-IPC proportional controller gain Kp

Y_IPC_KI
[Float] Yaw-by-IPC integral controller gain Ki

Y_IPC_omegaLP
[Float, rad/s.] Low-pass filter corner frequency for the Yaw-by-IPC controller to filtering the yaw alignment error

Y_IPC_zetaLP
[Float] Low-pass filter damping factor for the Yaw-by-IPC controller to filtering the yaw alignment error.

Y_MErrSet
[Float, rad] Yaw alignment error, set point

Y_omegaLPFast
[Float, rad/s] Corner frequency fast low pass filter, 1.0

Y_omegaLPSlow
[Float, rad/s] Corner frequency slow low pass filter, 1/60

Y_Rate
[Float, rad/s] Yaw rate

FA_KI
[Float, rad s/m] Integral gain for the fore-aft tower damper controller, -1 = off / >0 = on

FA_HPFCornerFreq
[Float, rad/s] Corner frequency (-3dB point) in the high-pass filter on the fore- aft acceleration signal

FA_IntSat
[Float, rad] Integrator saturation (maximum signal amplitude contribution to pitch from FA damper)

PS_BldPitchMin_N
[Float] Number of values in minimum blade pitch lookup table (should equal number of values in PS_WindSpeeds and PS_BldPitchMin)

PS_WindSpeeds
[Array of Floats] Wind speeds corresponding to minimum blade pitch angles

PS_BldPitchMin
[Array of Floats] Minimum blade pitch angles

SD_MaxPit
[Float, rad] Maximum blade pitch angle to initiate shutdown

SD_CornerFreq
[Float, rad/s] Cutoff Frequency for first order low-pass filter for blade pitch angle

Fl_Kp
[Float, s] Nacelle velocity proportional feedback gain

Flp_Angle
[Float, rad] Initial or steady state flap angle

Flp_Kp
[Float, s] Blade root bending moment proportional gain for flap control

Flp_Ki
[Float] Flap displacement integral gain for flap control

Flp_MaxPit
[Float, rad] Maximum (and minimum) flap pitch angle

OL_Filename
[String] Input file with open loop timeseries (absolute path or relative to this file)

Ind_Breakpoint
[Float] The column in OL_Filename that contains the breakpoint (time if OL_Mode = 1)

Ind_BldPitch
[Float] The column in OL_Filename that contains the blade pitch input in rad

Ind_GenTq
[Float] The column in OL_Filename that contains the generator torque in Nm

Ind_YawRate
[Float] The column in OL_Filename that contains the generator torque in Nm

DLL_FileName
[String] Name/location of the dynamic library {.dll [Windows] or .so [Linux]} in the Bladed-DLL format

Default = unused

DLL_InFile
[String] Name of input file sent to the DLL

Default = unused

DLL_ProcName
[String] Name of procedure in DLL to be called

Default = DISCON

PF_Offsets
[Array of Floats] Pitch angle offsets for each blade (array with length of 3)

CC_Group_N
[Float] Number of cable control groups

Default = 0

CC_GroupIndex
[Array of Floats] First index for cable control group, should correspond to deltaL

Default = [0]

CC_ActTau
[Float] Time constant for line actuator [s]

Default = 20

StC_Group_N
[Float] Number of cable control groups

Default = 0

StC_GroupIndex
[Array of Floats] First index for structural control group, options specified in ServoDyn summary output

Default = [0]

4.6.4 linmodel_tuning

Inputs used for tuning ROSCO using linear (level 2) models

type
[String from, ['none', 'robust', 'simulation']] Type of level 2 based tuning - robust gain scheduling (robust) or simulation based optimization (simulation)

Default = none

linfile_path
[String] Path to OpenFAST linearization (.lin) files, if they exist

Default = none

lintune_outpath
[String] Path for outputs from linear model based tuning

Default = lintune_outfiles

load_parallel
[Boolean] Load linearization files in parallel (True/False)

Default = False

stability_margin
[Float or Array of Floats] Desired maximum stability margin

Default = 0.1

4.7 Running Bladed simulations with ROSCO controller

ROSCO controller can be used with Bladed.

ROSCO dll must be built to 32bit windows version. Most pre-built discon.dlls in ROSCO github are 64bit, so you may need to build from source.

Configuration in Bladed is as follows:

4.7.1 Bladed versions 4.6 to current (4.12)

In the Bladed External Controller dialog, fill in the fields as follows:

- *'Time step'* value non-critical as ROSCO adapts to whatever value is specified. Suggest 10ms.
- *'Additional Controller Parameters'* - copy all text from DISCON.IN for the relevant turbine and paste in to this field.

  Notes:

  – This must be the topmost field called 'Additional Controller Parameters', not the field with the same name lower down in external controller 1 settings.

  – Do not add, remove or re-order any lines in this text as this will break ROSCO parsing of the content.

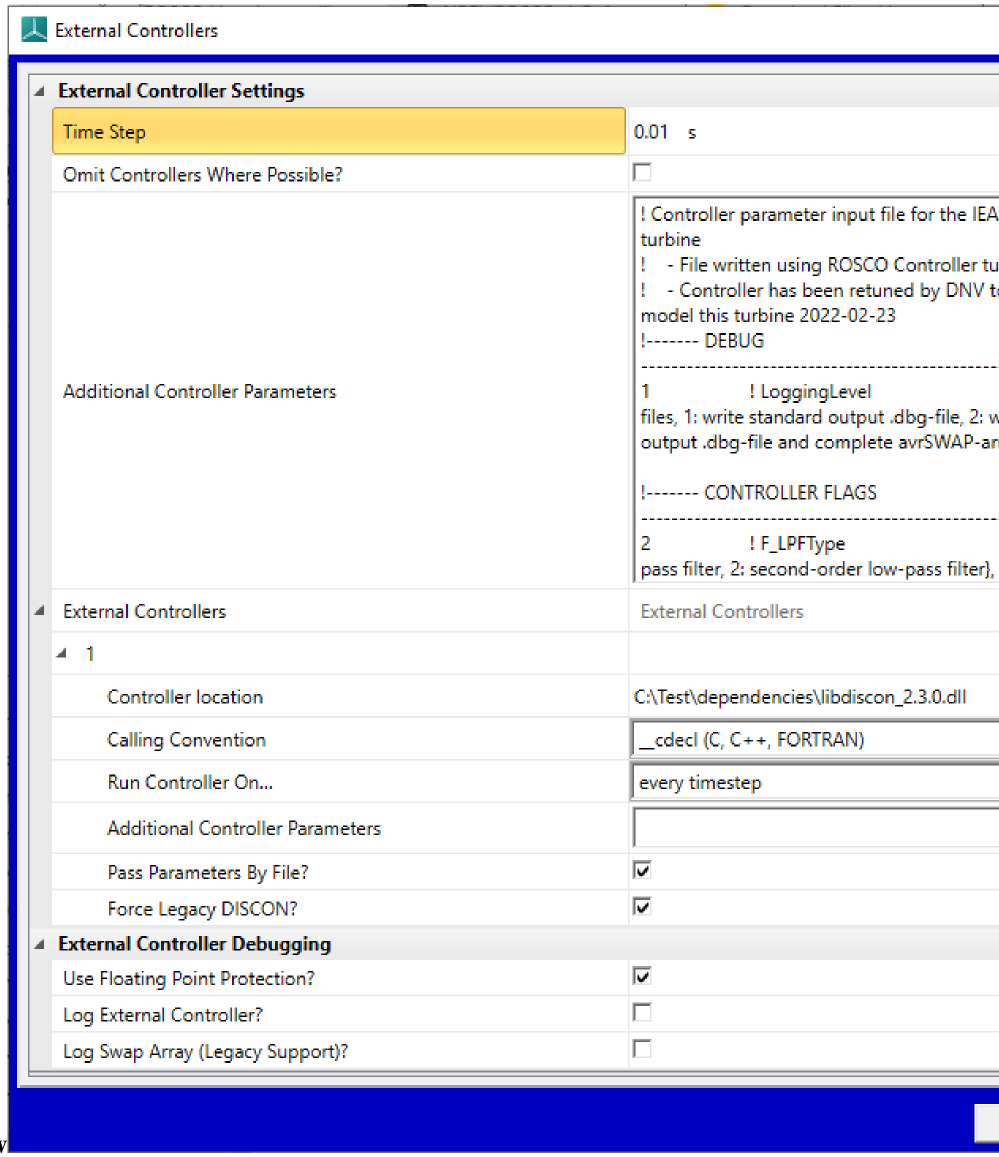
  – Any paths such as PerfFileName must be absolute (eg 'C:ROSCOconfig.txt', not '..config.txt') for use with Bladed.
- Add an external controller (click "+") and set
  – *'Controller location'* - path to the ROSCO libdiscon_win32.dll

  – *'Calling convention'* - __cdecl

  – *'Additional Controller Parameters'* - blank

  – *'Pass parameters by file'* - must be ticked (this instructs Bladed to create a DISCON.IN file at runtime with the text from the Additional Controller Parameters window, and ROSCO reads from this file)

  – *'Force legacy discon'* - ticked



Example setup shown in the image below

4.7.2 Bladed 4.5 & earlier

In External Controller dialog,

- *'Communication interval'* - ROSCO adapts to whatever value is specified. Suggest 10ms.
- *'Controller code'* - path to the ROSCO libdiscon_win32.dll
- *'Calling convention'* - __cdecl
- *'External Controller data'* - copy the configuration text from DISCON.IN for the relevant turbine and paste in to this field.

  Notes:

  – Do not add, remove or re-order any lines in this text as this will break ROSCO parsing of the content

  – Any paths such as PerfFileName must be absolute (eg 'C:ROSCOconfig.txt', not '..config.txt') for use with Bladed.

Troubleshooting (all Bladed versions)

Most error messages from ROSCO are not passed to the Bladed GUI for display or logging. They will be visible only in the transient DOS window that appears while a Bladed simulation is running. If there is a problem you will likely see only 'simulation terminated unexpectedly' in Bladed UI. To view error messages from ROSCO you will need to run Bladed from the command line. This will ensure that the console window where errors are displayed remains open to view the error message.

Instructions to run Bladed from the command line are available here on the Bladed Knowledge Base