# ROSCO

*Release 2.9.4*

**Nikhar J. Abbas, Daniel S. Zalkind**

**Oct 04, 2024**

# CONTENTS

**Version**
2.9.4

**Date**
Oct 04, 2024

NREL's "Reference Open Source Controller" (ROSCO) is a reference controller framework that facilitates design and implementation of wind turbine and wind farm controllers for fixed and floating offshore wind turbines.

ROSCO frameworks includes a large set of available controllers and advanced functionalities that can be combined in a modular fashion based on the intended application and can be easily adapted to a wide variety wind turbines. For example, ROSCO can be used to design turbine yaw controller along with an individual blade pitch controller with floating platform feedback for an offshore turbine while simulating a pitch actuator fault and running a user-defined torque controller.

ROSCO provides a single framework for designing controllers for onshore and offshore turbines of varying sizes. It can be used to run representative dynamic simulations using OpenFAST. This helps researchers perform 'apples-to-apples' comparison of controller capabilities across turbines. Control engineers can also design their own controllers and compare them with reference controller design using ROSCO for existing and new turbines. ROSCO has been used to provide reference controllers for many recent reference turbines including the IEA 3.4-MW , IEA 10-MW , IEA 15-MW and the upcoming IEA 22-MW turbines.

The ROSCO framework also includes a python based toolbox that primarily enables tuning the controllers. The tuning process is extemely simple where only a tuning parameters need to be provided. It is not necessary to run aeroelastic simulations or provide linearized state-space models to tune the controller to tune the controllers. The toolbox has other capabilities like simple 1-DOF turbine simulations for quick controller capability verifications, linear model analysis, and parsing of input and output files.

Source code for ROSCO toolset can be found in this github repository and it can be installed following the instructions provided in *Installing ROSCO toolset*.

**Documentation Directory**

CHAPTER

ONE

# STANDARD ROSCO WORKFLOW

Fig. 1.1: ROSCO toolchain general workflow

Fig. 1.1 shows the general workflow for the ROSCO tool-chain with OpenFAST. For the standard use case in OpenFAST (or similar), ROSCO controller needs to be compiled. The controller is a fortran based module that follows the bladed-style control interface. Compiling the controller ouputs a dynamic-link library (or equivalent) called `libdiscon.dll` for windows, `libdiscon.so` for linux and, `libdiscon.dylib` for mac-os. Instructions for the compilation are provided in *Installing ROSCO toolset*. Once the controller is compiled the turbine simulation tool must point to the compiled library. In OpenFAST, this is ensured by changing the `DLL_FileName` parameter in the ServoDyn input file. This step enables communication between the ROSCO controller and OpenFAST.

The compiled ROSCO controller library requires an input file (generally called `DISCON.IN`). It stores several flags and parameters needed by the controller and is read by the compiled dynamic-link library. Several different `DISCON.IN` files, for varous turbines and controller tunings, can use the same dynamic-link library. In OpenFAST, the `DLL_InFile` parameter in the ServoDyn input file determines the desired input file.

The ROSCO toolbox is used to tune the ROSCO controller and generate a `DISCON.IN` input file. To tune the controller, ROSCO toolbox needs the OpenFAST model of the turbine and some user inputs in the form of a `tuning.yaml` file. The functionality of ROSCO toolset can be best understood by following the set of included example scripts in *ROSCO Examples*. ROSCO toolset can be installed using the instructions provided in *Installing ROSCO toolset*.

CHAPTER

TWO

# INSTALLING ROSCO TOOLSET

ROSCO toolsets can be utilized either to run an existing controller or to design and tune a controller from scratch. We recommend using the instructions provided in the *Complete ROSCO Installation* to install the full ROSCO toolset. This allows for full use of the provided functionalities including the controller and toolbox to facilitate controller tuning. However, if only the ROSCO binary is needed (to run an existing controller, for example), then users should follow the instructions provided in *Installing only the ROSCO controller*

## 2.1 Complete ROSCO Installation

Steps for the installation of the complete rosco toolset are:

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=3.10 # (Create a new environment named "rosco-env
↪" that contains Python 3.8)
conda activate rosco-env # (Activate your "rosco-env" environment)

# Windows users sometimes get an error related to the SSL configuration; in this case,␣
↪use
# Be sure to execute commands in an anaconda terminal rather than the windows command␣
↪prompt
conda config --set ssl_verify no

# Install necessary compilers
conda install -y m2w64-toolchain libpython  # windows
conda install compilers                     # unix


# If you intend to use ZeroMQ
brew install zeromq  # mac
sudo apt install libzmq3-dev libzmq5 libczmq-dev libczmq4  # linux
```

2. Clone and Install the ROSCO toolbox with ROSCO controller

```
git clone https://github.com/NREL/ROSCO.git
cd ROSCO
pip install -e . --no-deps
```

This step creates the rosco controller binary (`libdiscon.so` (Linux), `libdiscon.dylib` (Mac), or `libdisscon.dll` (Windows)`) in the directory `ROSCO/rosco/lib` and installs the python toolbox in the conda environment in the develop mode.

3. If for some reason the pip-based installation does not work, the conda environment can be created using

```
conda env update --file enviroment.yml
pip install -e . --no-deps
```

## 2.2 Installing only the ROSCO controller

Table 2.1 provides an overview of the primary methods available for installing only the ROSCO controller binary.

Table 2.1: 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 binary without working through the compilation procedures. |
| *Anaconda Download* | Best for users who just want to use the controller binary but prefer to download using the Anaconda package manager. |
| *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. |

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.

### 2.2.1 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).

### 2.2.2 Anaconda Download

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=3.10
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.

### 2.2.3 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/controller
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).

CHAPTER

# THREE

# ROSCO EXAMPLES

Methods for reading turbine models, generating the control parameters of a `DISCON.IN`: file, and running aeroelastic simulations to test controllers 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`.

## 3.1 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

## 3.2 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

## 3.3 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.

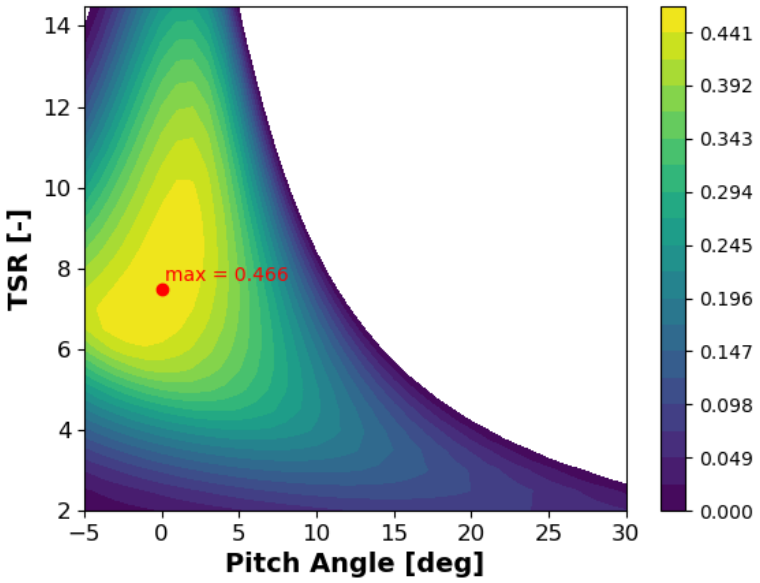
## 3.4 List of Examples

A complete list of examples is given below:

### 3.4.1 01_turbine_model

Interact with a ROSCO turbine model:

- Read .yaml input file
- Load an openfast turbine model
- Read text file with rotor performance properties (Cp, Ct, and Cq surface)
- Print some basic turbine properties
- Save the turbine as a pickle
- Plot the Cp surface




Note: Uses the NREL 5MW included in the Test Cases and is a part of the OpenFAST distribution

### 3.4.2 02_ccblade

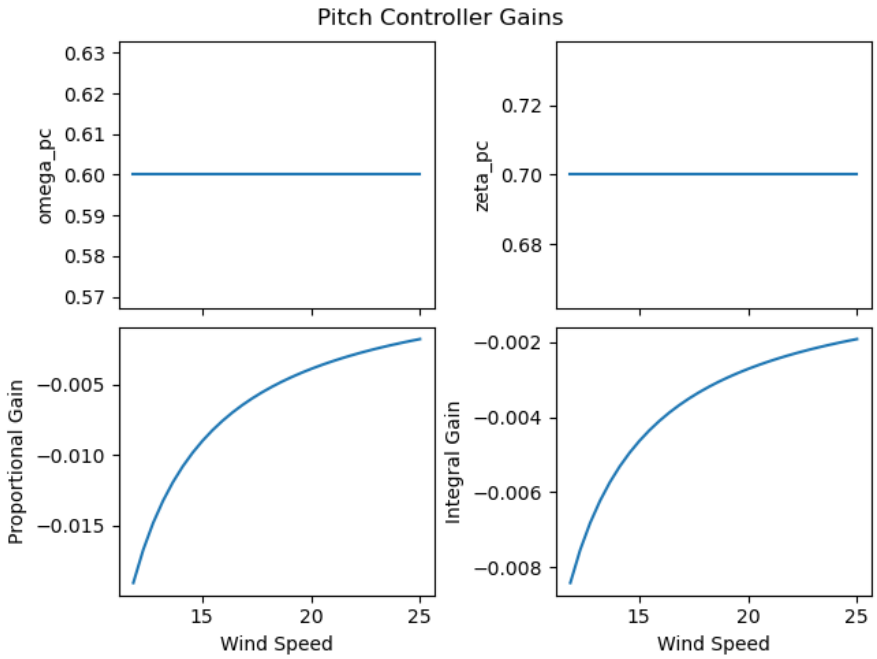
Run CCblade, save a rotor performance text file. In this example:

- Read .yaml input file
- Load an openfast turbine model
- Run ccblade to get rotor performance properties
- Write a text file (`'02_Cp_Ct_Cq.Ex03.txt'`) with rotor performance properties

### 3.4.3 03_tune_controller

Load a turbine model and tune the controller In this example:

- Read a .yaml file
- Create a ROSCO turbine object fromt he OpenFAST model
- Tune a ROSCO controller object
- Write a controller input file (`'03_DISCON.IN'`)
- Plot gain schedule (`PC_GS_KP` and `PC_GS_KI` versus `PS_GS_angles`):

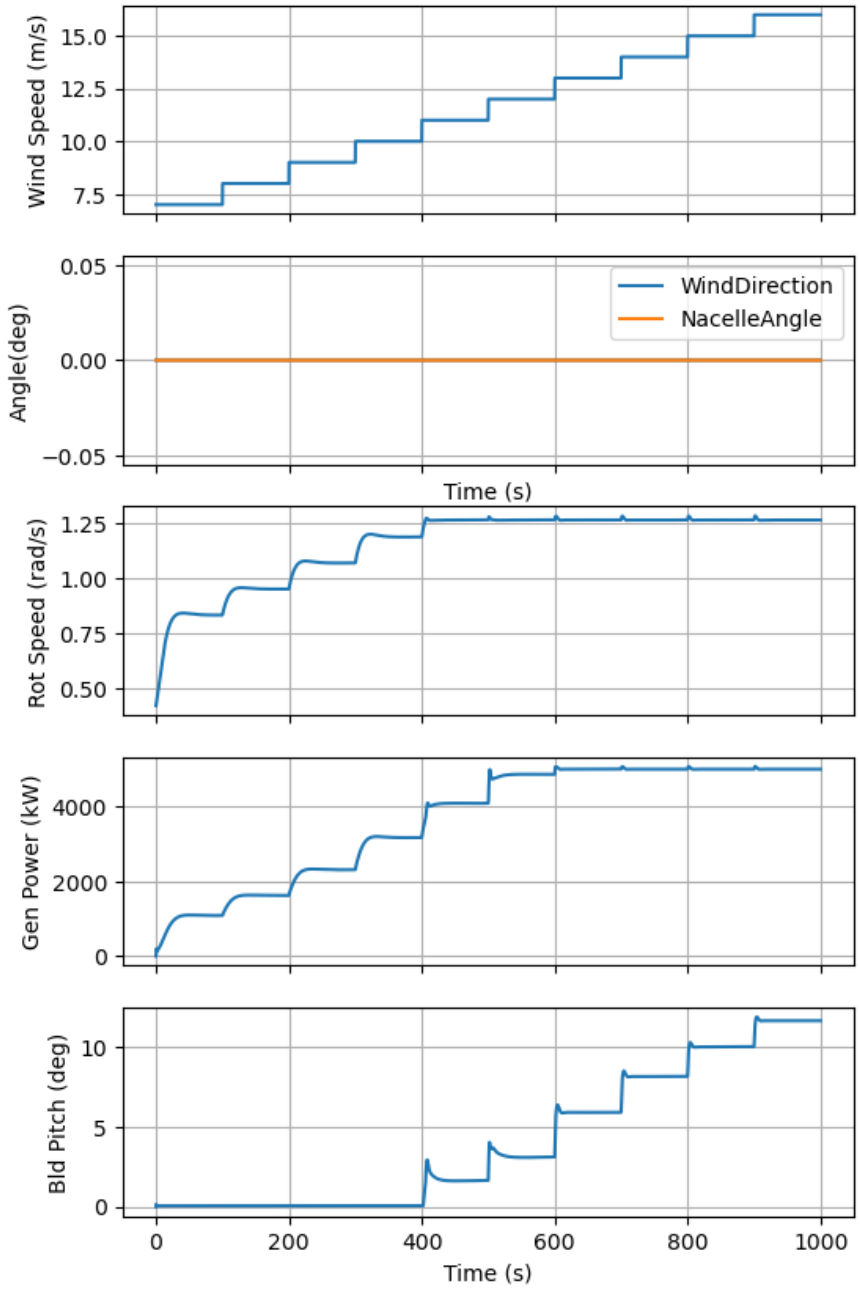


### 3.4.4 04_simple_sim

Demonstrate the simple 1-DOF wind turbine simulator with ROSCO

In this example:

- Load turbine from saved pickle and tune a ROSCO controller
- Run and plot a step wind simulation using 1-DOF model in `rosco.toolbox.sim` and the ROSCO dynamic library




Notes:

- You must have a compiled controller in ROSCO/rosco/lib/, and properly point to it using the *lib_name* variable.

- Using wind speed estimators in this simple simulation is known to cause problems. We suggest using WE_Mode = 0 in the DISCON.IN or increasing sampling rate of simulation as workarounds.
- The simple simulation is run twice to check that arrays are deallocated properly.

### 3.4.5 05_openfast_sim

Load a turbine, tune a controller, and run an OpenFAST simulation. In this example:

- Load a turbine from OpenFAST
- Tune a controller
- Run an OpenFAST simulation

Note

- You must have a compiled controller in ROSCO/rosco/lib/

### 3.4.6 06_peak_shavings

Load saved turbine, tune controller, plot minimum pitch schedule In this example:

- Load a yaml file
- Load a turbine from openfast
- Tune a controller
- Plot minimum pitch schedule

### 3.4.7 07_openfast_outputs

Plot some OpenFAST output data In this example:

- Load openfast output data
- Trim the time series
- Plot some available channels

Note: need to run openfast model in 'Test_Cases/5MW_Land_DLL_WTurb/' to plot

### 3.4.8 08_run_turbsim

Run TurbSim to create wind field binary In this example:

- Leverage the run_openfast functionality to compile a turbsim binary

### 3.4.9 09_distributed_aero

Tune a controller for distributed aerodynamic control In this example:

- Read .yaml input file
- Load an openfast turbine model
- Read text file with rotor performance properties
- Load blade information
- Tune controller with flap actuator

Note

- You will need a turbine model with DAC capabilites in order to run this. The curious user can contact Nikhar Abbas (nikhar.abbas@nrel.gov) for available models, if they do not have any themselves.

### 3.4.10 10_linear_params

Load a turbine, tune a controller, export linear model In this example:

- Load a turbine from OpenFAST
- Tune a controller
- Use tuning parameters to export linear model

### 3.4.11 11_robust_tuning

Controller tuning to satisfy a robustness criteria

Note that this example necessitates the mbc3 through either pyFAST or WEIS pyFAST is the easiest to install by cloning https://github.com/OpenFAST/openfast_toolbox and running `python setup.py develop` from your conda environment

In this example:

- setup ROSCO's robust tuning methods for the IEA15MW on the UMaine Semi-sub
- run a the standard tuning method to find k_float
- run robust tuning to find omega_pc schedule satisfy a prescribed stability margin
- Tune ROSCO's pitch controller using omega_pc schedule
- Plot gain schedule

The example is put in a function call to show the ability to load linear models in parallel

### 3.4.12 12_tune_ipc

Load a turbine, tune a controller with IPC In this example:

- Load a turbine from OpenFAST
- Tune a controller with IPC
- Run simple simulation with open loop control

### 3.4.13 14_open_loop_control

Load a turbine, tune a controller with open loop control commands In this example:

- Load a turbine from OpenFAST
- Tune a controller
- Write open loop inputs
- Run simple simulation with open loop control

### 3.4.14 15_pass_through

Use the runFAST scripts to set up an example, use pass through in yaml In this example:

- use run_FAST_ROSCO class to set up a test case

### 3.4.15 16_external_dll

Run openfast with ROSCO and external control interface IEA-15MW will call NREL-5MW controller and read control inputs

### 3.4.16 17a_zeromq_simple

Run ROSCO using the ROSCO toolbox control interface and execute communication with ZeroMQ.

A demonstrator for ZeroMQ communication. Instead of using ROSCO with with control interface, one could call ROSCO from OpenFAST, and communicate with ZeroMQ through that. this_dir

### 3.4.17 17b_zeromq_multi_openfast

Run multiple openfast simulations and execute communication with ZeroMQ.

### 3.4.18 18_pitch_offsets

Run openfast with ROSCO and pitch offset faults Set up and run simulation with pitch offsets, check outputs

### 3.4.19 19_update_discon_version

Test and demonstrate update_discon_version() function for converting an old ROSCO input to the current version

### 3.4.20 20_active_wake_control

Run openfast with ROSCO and active wake control Set up and run simulation with AWC, check outputs Active wake control (AWC) with blade pitching is implemented in this example with two approaches as detailed below:

### 3.4.21 21_optional_inputse_discon_version

Test and demonstrate update_discon_version() function for converting an old ROSCO input to the current version

### 3.4.22 22_cable_control

Run openfast with ROSCO and cable control Set up and run simulation with pitch offsets, check outputs

ROSCO currently supports user-defined hooks for cable control actuation, if CC_Mode = 1. The control logic can be determined in Controllers.f90 with the CableControl subroutine. The CableControl subroutine takes an array of CC_DesiredL (length) equal to the ChannelIDs set in MoorDyn and determines the length and change in length needed for MoorDyn using a 2nd order actuator model (CC_ActTau). In the DISCON input, users must specify CC_GroupIndex relating to the deltaL of each control ChannelID. These indices can be found in the ServoDyn summary file (*SrvD.sum)

In the example below (and hard-coded in ROSCO) a step change of -10 m on line 1 is applied at 50 sec.

### 3.4.23 23_structural_control

Run openfast with ROSCO and structural control Set up and run simulation with pitch offsets, check outputs

ROSCO currently supports user-defined hooks for structural control control actuation, if StC_Mode = 1. The control logic can be determined in Controllers.f90 with the StructrualControl subroutine. In the DISCON input, users must specify StC_GroupIndex relating to the control ChannelID. These indices can be found in the ServoDyn summary file (*SrvD.sum)

In the example below, we implement a smooth step change mimicing the exchange of ballast from the upwind column to the down wind columns

OpenFAST v3.5.0 is required to run this example

### 3.4.24 24_floating_feedback

Run openfast with ROSCO and all the floating feedback methods Floating feedback methods available in ROSCO/ROSCO_Toolbox

1. Automated tuning, constant for all wind speeds
2. Automated tuning, varies with wind speed
3. Direct tuning, constant for all wind speeds
4. Direct tuning, varies with wind speeds

### 3.4.25 25_rotor_position_control

Run ROSCO with rotor position control Run a steady simulation, use the azimuth output as an input to the next steady simulation, with different ICs

### 3.4.26 26_marine_hydro

Run MHK turbine in OpenFAST with ROSCO torque controller

### 3.4.27 27_power_ref_control

Run openfast with ROSCO and cable control Demonstrate a simulation with a generator reference speed that changes with estimated wind speed Set reference rotor speed as a function of wind speed (estimate in ROSCO)

### 3.4.28 28_tower_resonance

Demonstrate tower resonance avoidance controller Set up and run simulation with tower resonance avoidance

CHAPTER

FOUR

# ROSCO STRUCTURE: CONTROLLER

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

## 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; it is procedurally generated from `rosco_registry`
- `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.
- `ExtControl.f90` contains subroutines for calling external dynamic libraries
- `ROSCO_Helpers.f90` contains subroutines for file I/O and other helpful routines, borrowed heavily from NWTC.IO in OpenFAST
- `ROSCO_IO.f90` is procedurally generated using the `rosco_registry` for writing debug and checkpoint files

## 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 4.1: 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 |
| CONTROLLEI 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, [-] |
| | 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 |

continues on next page

Table 4.1 – continued from previous page

| Primary Section | Variable | Type | Description |
|---|---|---|---|
| | 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] |
| | 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]. |

continues on next page

Table 4.1 – continued from previous page

| Primary Section | Variable | Type | Description |
|---|---|---|---|
| | 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) |
| | 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] |

continues on next page

Table 4.1 – continued from previous page

| Primary Section | Variable | Type | Description |
|---|---|---|---|
| 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. |
| | 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] |

continues on next page

Table 4.1 – continued from previous page

| Primary Section | Variable | Type | Description |
|---|---|---|---|
| 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] |

CHAPTER

FIVE

# ROSCO STRUCTURE: TOOLBOX

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

## 5.1 ROSCO Toolbox File Structure

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

### 5.1.1 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.

### 5.1.2 Examples

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

### 5.1.3 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.

### 5.1.4 Examples/Test_Cases

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

### 5.1.5 Examples/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.

## 5.2 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*.

### 5.2.1 Matlab_Toolbox

A simulink implementation of the ROSCO controller is included in the Matlab Toolbox. Some requisite MATLAB utility scripts are also included. These scripts are not maintained by NREL as of ROSCO v2.5.0. The Simulink controller there should not be used and referenenced as "ROSCO" because it has never been validated against the official ROSCO dynamic library and has drifted away from the official implementation. We will leave the implementation in place to be used only as a learning tool.

CHAPTER

SIX

# 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.

## 6.1 2.8.0 to 2.9.0

**Flag to use exteneded Bladed Interface**

- Set *Ext_Interface* to 1 to use the extened bladed interface with OpenFAST v3.5.0 and greater

**Gain scheduling of floating feedback**

- The floating feedback gain can be scheduled on the low pass filtered wind speed signal. Note that Fl_Kp can now be an array.

**Rotor position tracking**

- Control the azimuth position of the rotor with *OL_Mode* of 2 using a PID torque controller with gains defined by *RP_Gains*.
- Control all three blade pitch inputs in open loop

**New torque control mode settings**

- VS_ControlMode determines how the generator speed set point is determined: using the WSE (mode 2) or (P/K)^(1/3) (mode 3). The power signal in mode 3 is filtered using *VS_PwrFiltF*.
- VS_ConstPower determines whether constant power is used (0 is constant torque, 1 is constant power)

**Multiple notch filters**

- Users can list any number of notch filters and apply them to either the generator speed and/or tower top acceleration signal based on their index

**Power reference control via generator speed set points**

- With this feature, enabled with *PRC_Mode*, a user can prescribe a set of generator speed set points (*PRC_GenSpeeds*) vs. the estimated wind speed (*PRC_WindSpeeds*), which can be used to avoid certain natural frequencies or implement a soft cut-out scheme.
- A low pass filter with frequency *PRC_LPF_Freq* is used to filter the wind speed estimate. A lower value increases the stability of the generator speed reference signal.

**ZeroMQ Interface**

- Each turbine is assigned a *ZMQ_ID* by the controller, which is tracked by a farm-level controller

**Tower resonance avoidance**

- When *TRA_Mode* is 1, change the torque control generator speed setpoint to avoid TRA_ExclSpeed +/- TRA_ExclBand.
- The set point is changed at a slow rate *TRA_RateLimit* to avoid generator power spikes. *VS_RefSpd*/100 is recommended.

Removed in ROSCO develop

| Line | Input Name | Example Value |
|---|---|---|
| 11 | F_NotchTy | 2 ! F_NotchType - 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} |
| 35 | F_NotchCo | 3.35500 ! F_NotchCornerFreq - Natural frequency of the notch filter, [rad/s] |
| 36 | F_NotchBe | 0.000000 0.250000 ! F_NotchBetaNumDen - Two notch damping values (numerator and denominator, resp) - determines the width and depth of the notch, [-] |

| New in ROSCO develop<br>Line | Input Name | Example Value |
|---|---|---|
| 7 | Ext_Interface | 1 ! Ext_Interface - (0 - use standard bladed interface, 1 - Use the extened DLL interface introduced in OpenFAST 3.5.0.) |
| 14 | VS_ConstPow | 0 ! VS_ConstPower - Do constant power torque control, where above rated torque varies, 0 for constant torque} |
| 18 | PRC_Mode | 0 ! PRC_Mode - Power reference tracking mode{0: use standard rotor speed set points, 1: use PRC rotor speed setpoints} |
| 38 | F_NumNotch | 1 ! F_NumNotchFilts - Number of notch filters placed on sensors |
| 39 | F_NotchFreqs | 3.3550 ! F_NotchFreqs - Natural frequency of the notch filters. Array with length F_NumNotchFilts |
| 40 | F_NotchBetal | 0.0000 ! F_NotchBetaNum - Damping value of numerator (determines the width of notch). Array with length F_NumNotchFilts, [-] |
| 41 | F_NotchBetal | 0.2500 ! F_NotchBetaDen - Damping value of denominator (determines the depth of notch). Array with length F_NumNotchFilts, [-] |
| 42 | F_GenSpdNo | 0 ! F_GenSpdNotch_N - Number of notch filters on generator speed |
| 43 | F_GenSpdNo | 0 ! F_GenSpdNotch_Ind - Indices of notch filters on generator speed |
| 44 | F_TwrTopNot | 1 ! F_TwrTopNotch_N - Number of notch filters on tower top acceleration signal |
| 45 | F_TwrTopNot | 1 ! F_TwrTopNotch_Ind - Indices of notch filters on tower top acceleration signal |
| 92 | VS_PwrFiltF | 0.3140 ! VS_PwrFiltF - Low pass filter on power used to determine generator speed set point. Only used in VS_ControlMode = 3. |
| 98 | PRC_Section | !——- POWER REFERENCE TRACKING ——————————— |
| 99 | PRC_n | 2 ! PRC_n - Number of elements in PRC_WindSpeeds and PRC_GenSpeeds array |
| 100 | PRC_LPF_Fr | 0.07854 ! PRC_LPF_Freq - Frequency of the low pass filter on the wind speed estimate used to set PRC_GenSpeeds [rad/s] |
| 101 | PRC_WindSp | 3.0000 25.0000 ! PRC_WindSpeeds - Array of wind speeds used in rotor speed vs. wind speed lookup table [m/s] |
| 102 | PRC_GenSpe | 0.7917 0.7917 ! PRC_GenSpeeds - Array of generator speeds corresponding to PRC_WindSpeeds [rad/s] |
| 103 | Empty Line | |
| 128 | TRA_ExclSp | 0.00000 ! TRA_ExclSpeed - Rotor speed for exclusion [LSS, rad/s] |
| 129 | TRA_ExclBa | 0.00000 ! TRA_ExclBand - Size of the rotor frequency exclusion band [LSS, rad/s]. Torque controller reference will be TRA_ExclSpeed +/- TRA_ExlBand/2 |
| 130 | TRA_RateLir | 0.00000e+00 ! TRA_RateLimit - Rate limit of change in rotor speed reference [LSS, rad/s]. Suggested to be VS_RefSpd/100. |
| 145 | Fl_n | 1 ! Fl_n - Number of Fl_Kp gains in gain scheduling, optional with default of 1 |
| 147 | Fl_U | 0.0000 ! Fl_U - Wind speeds for scheduling Fl_Kp, optional if Fl_Kp is single value [m/s] |
| 161 | Ind_Azimuth | 0 ! Ind_Azimuth - The column in OL_Filename that contains the desired azimuth position in rad (used if OL_Mode = 2) |
| 162 | RP_Gains | 0.0000 0.0000 0.0000 0.0000 ! RP_Gains - PID gains and Tf of derivative for rotor position control (used if OL_Mode = 2) |
| 186 | ZMQ_ID | 0 ! ZMQ_ID - Integer identifier of turbine |

| Changed in ROSCO develop | | |
|---|---|---|
| Line | Input Name | Example Value |
| 12 | VS_Cc | 2 ! VS_ControlMode - Generator torque control mode in above rated conditions (0- no torque control, 1- k*omega^2 with PI transitions, 2- WSE TSR Tracking, 3- Power-based TSR Tracking)}126 OL_mode 0 ! OL_Mode - Open loop control mode {0: no open loop control, 1: open loop control vs. time, 2: rotor position control} |
| 125 | Twr_S | !——- TOWER CONTROL ———————————————————— |
| 141 | Fl_Kp | 0.0000 ! Fl_Kp - Nacelle velocity proportional feedback gain [s] |
| 153 | Ind_Bl | 0 0 0 ! Ind_BldPitch - The columns in OL_Filename that contains the blade pitch (1,2,3) inputs in rad [array] |

## 6.2 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<br>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_Groupl | 2818 2838 2858 ! StC_GroupIndex - First index for structural control group, options specified in ServoDyn summary output |

## 6.3 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 |
| 22 | PA_Moc | 0 ! PA_Mode - Pitch actuator mode {0 - not used, 1 - first order filter, 2 - second order filter} |
| 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 | |

## 6.4 2.5.0 to develop

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_zetaL | 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_omegaLPS | 0.10470 ! Y_omegaLPSlow - Corner frequency slow low pass filter, 1/60 [rad/s] |

## 6.5 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 | Input 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_WECorner | 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_Breakpoi | 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 |

CHAPTER

SEVEN

# ROSCO_TOOLBOX TUNING .YAML

Definition of inputs for ROSCO tuning procedure

toolbox_schema

## 7.1 path_params

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

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

**rotor_performance_filename**
[String] Filename for rotor performance text file (if it has been generated by ccblade already), relative to directory of this yaml

## 7.2 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

**reynolds_ref**
[Float] Reynolds number near rated speeds, used to interpolate airfoils, if provided

*Default* = 0

*Minimum* = 0

## 7.3 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- no torque control, 1- k*omega^2 with PI transitions, 2- WSE TSR Tracking, 3- Power-based TSR Tracking)

*Default* = 2

*Minimum* = 0 *Maximum* = 3

**VS_ConstPower**
[Float] Do constant power torque control, where above rated torque varies, 0 for constant torque

*Default* = 0

*Minimum* = 0 *Maximum* = 1

**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

**TRA_Mode**
[Float] Tower resonance avoidance mode (0- no tower resonsnace avoidance, 1- use torque control setpoints to avoid a specific frequency

*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

**ZMQ_UpdatePeriod**
[Float] Call ZeroMQ every [x] seconds, [s]

*Default* = 2

*Minimum* = 0

**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

**OL_Mode**
[Float] Open loop control mode {0- no open loop control, 1- open loop control}

*Default* = 0

*Minimum* = 0 *Maximum* = 2

**AWC_Mode**
[Float] Active wake control mode {0 - not used, 1 - SNL method, 2 - NREL method}

*Default* = 0

*Minimum* = 0 *Maximum* = 2

**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- Open loop control]

*Default* = 0

*Minimum* = 0 *Maximum* = 2

**StC_Mode**
[Float] Structural control mode [0- unused, 1- User defined, 2- Open loop control]

*Default* = 0

*Minimum* = 0 *Maximum* = 2

**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 or Array of Floats] Gain(s) of floating feedback control

**tune_Fl**
[Boolean] Whether to automatically tune Kp_float

*Default* = True

**U_Fl**
[Array of Floats or String or Float] List of wind speeds for tuning floating feedback, or "all" for all above-rated wind speeds

*Default* = []

**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

rgn2k_factor
: [Float] Factor on VS_Rgn2K to increase/decrease optimal torque control gain, default is 1. Sometimes environmental conditions or differences in BEM solvers necessitate this change.

*Default* = 1

*Minimum* = 0

### 7.3.1 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

### 7.3.2 open_loop

**flag**
[Boolean] Flag to use open loop control

*Default* = False

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

*Default* = unused

**Ind_Breakpoint**
[Float] Index (column, 1-indexed) of breakpoint (time) in open loop index

*Default* = 1

*Minimum* = 0

**Ind_BldPitch**
[Array of Floats] Indices (columns, 1-indexed) of pitch (1,2,3) inputs in open loop input

*Default* = [0, 0, 0]

*Minimum* = 0

**Ind_GenTq**
[Float] Index (column, 1-indexed) of generator torque in open loop input

*Default* = 0

*Minimum* = 0

**Ind_YawRate**
[Float] Index (column, 1-indexed) of nacelle yaw in open loop input

*Default* = 0

*Minimum* = 0

**Ind_Azimuth**
[Float] The column in OL_Filename that contains the desired azimuth position in rad (used if OL_Mode = 2)

*Default* = 0

**Ind_CableControl**
[Array of Floats] The column in OL_Filename that contains the cable control inputs in m

**Ind_StructControl**
[Array of Floats] The column in OL_Filename that contains the structural control inputs in various units

**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

### 7.3.3 DISCON

These are pass-through parameters for the DISCON.IN file. Use with caution. Do not set defaults in schema.

**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

**DT_Out**
[Float] Time step to output .dbg* files, or 0 to match sampling period of OpenFAST

*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

**VS_ControlMode**
[Float] Generator torque control mode in above rated conditions (0- no torque control, 1- k*omega^2 with PI transitions, 2- WSE TSR Tracking, 3- Power-based TSR Tracking)

*Minimum* = 0 *Maximum* = 3

**VS_ConstPower**
[Float] Do constant power torque control, where above rated torque varies

*Minimum* = 0 *Maximum* = 1

**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)

**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)

**OL_Mode**
[Float] Open loop control mode (0 - no open-loop control, 1 - direct open loop control, 2 - rotor position 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_NumNotchFilts**
[Float] Number of notch filters placed on sensors

**F_NotchFreqs**
[Array of Floats or Float, rad/s] Natural frequency of the notch filters. Array with length F_NumNotchFilts

**F_NotchBetaNum**
[Array of Floats or Float] Damping value of numerator (determines the width of notch). Array with length F_NumNotchFilts, [-]

**F_NotchBetaDen**
[Array of Floats or Float] Damping value of denominator (determines the depth of notch). Array with length F_NumNotchFilts, [-]

**F_GenSpdNotch_N**
[Float] Number of notch filters on generator speed

**F_TwrTopNotch_N**
[Float] Number of notch filters on tower top acceleration signal

**F_GenSpdNotch_Ind**
[Array of Floats or Float] Indices of notch filters on generator speed

**F_TwrTopNotch_Ind**
[Array of Floats or Float] Indices of notch filters on tower top acceleration signal

**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). Only used in VS_ControlMode = 1,3

**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. Only used in VS_ControlMode = 2.

**VS_PwrFiltF**
[Float, rad] Low pass filter on power used to determine generator speed set point. Only used in VS_ControlMode = 3.

*Default* = 0.314

**SS_VSGain**
[Float] Variable speed torque controller setpoint smoother gain

**SS_PCGain**
[Float] Collective pitch controller setpoint smoother gain

**PRC_Mode**
[Float] Power reference tracking mode, 0- use standard rotor speed set points, 1- use PRC rotor speed setpoints

**PRC_WindSpeeds**
[Array of Floats] Array of wind speeds used in rotor speed vs. wind speed lookup table [m/s]

**PRC_GenSpeeds**
[Array of Floats] Array of generator speeds corresponding to PRC_WindSpeeds [rad/s]

**PRC_LPF_Freq**
[Float] Frequency of the low pass filter on the wind speed estimate used to set PRC_GenSpeeds [rad/s]

*Default* = 0.078539

**PRC_n**
[Float] Number of elements in PRC_WindSpeeds and PRC_GenSpeeds array

**TRA_ExclSpeed**
[Float] Rotor speed for exclusion [LSS, rad/s]

*Default* = 0.0

*Minimum* = 0

**TRA_ExclBand**
[Float] Size of the rotor frequency exclusion band [LSS, rad/s]. Torque controller reference will be TRA_ExclSpeed +/- TRA_ExlBand/2

*Default* = 0.0

*Minimum* = 0

**TRA_RateLimit**
[Float] Rate limit of change in rotor speed reference [LSS, rad/s]. Suggested to be VS_RefSpd/400.

*Default* = 0.0

*Minimum* = 0

**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_n**
[Float, s] Number of Fl_Kp gains in gain scheduling, optional with default of 1

**Fl_Kp**
[Array of Floats] Nacelle velocity proportional feedback gain

**Fl_U**
[Array of Floats] Wind speeds for scheduling Fl_Kp, optional if Fl_Kp is single value [m/s]

**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 > 0)

**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

**Ind_Azimuth**
[Float] The column in OL_Filename that contains the desired azimuth position in rad (used if OL_Mode = 2)

**RP_Gains**
[Array of Floats] PID gains and Tf of derivative for rotor position control (used if OL_Mode = 2)

*Default* = [0, 0, 0, 0]

**Ind_CableControl**
[Array of Floats] The column in OL_Filename that contains the cable control inputs in m

**Ind_StructControl**
[Array of Floats] The column in OL_Filename that contains the structural control inputs in various units

**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)

*Default* = [0, 0, 0]

**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]

**AWC_Mode**
[Float] Active wake control mode {0 - not used, 1 - complex number method, 2 - Coleman transformation method}

*Default* = 0

*Minimum* = 0 *Maximum* = 2

**AWC_NumModes**
[Float, rad] Number of AWC modes

*Default* = 1

**AWC_n**
[Array of Floats] AWC azimuthal number (only used in complex number method)

*Default* = [1]

**AWC_harmonic**
[Array of Integers] AWC Coleman transform harmonic (only used in Coleman transform method)

*Default* = [1]

**AWC_freq**
[Array of Floats] AWC frequency [Hz]

*Default* = [0.05]

**AWC_amp**
[Array of Floats] AWC amplitude [deg]

*Default* = [1.0]

**AWC_clockangle**
[Array of Floats] AWC clock angle [deg]

*Default* = [0]

**ZMQ_CommAddress**
[String] Communication address for ZMQ server, (e.g. "tcp://localhost:5555")

*Default* = tcp://localhost:5555

**ZMQ_UpdatePeriod**
[Float] Update period at zmq interface to send measurements and wait for setpoint [sec.]

*Default* = 1.0

**ZMQ_ID**
[Float] Integer identifier of turbine

*Default* = 0

## 7.4 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

CHAPTER

EIGHT

# HOW TO CONTRIBUTE CODE TO ROSCO

ROSCO is an open-source tool, thus we welcome users to submit additions or fixes to the code to make it better for everybody. When adding new features to ROSCO, we strive to make them as generic as possible, so they can be applied to a variety of turbines. We also hope that new features are also representative of methods in most OEM turbines.

## 8.1 Issues

If you have an issue with ROSCO, a bug to report, or a feature to request, please submit an issue on the GitHub repository. This lets other users know about the issue. If you are comfortable fixing the issue, please do so and submit a pull request.

## 8.2 Documentation

When you add or modify code, make sure to provide relevant documentation that explains the new code. This should be done in code via comments and also in the Sphinx documentation if you add a new feature or capability. Look at the .rst files in the `docs` section of the repo or click on `view source` on any of the doc pages to see some examples.

The best place to add documentation for your new feature is in an example of the new feature. We are planning to incorporate docstrings from the examples into these docs.

To build the documentation locally, first

```
conda install -y cmake compilers sphinx sphinxcontrib-bibtex
conda install -y sphinx_rtd_theme>=1.3
```

Then

```
sphinx-build . ./_build/
```

## 8.3 Testing

ROSCO tests its various features through the Examples. Ideally, each new feature would have an associated example. Most examples are set up to simply run the example and check for simulation failures. Some good examples check that the functionality is occurring properly.

An automated testing procedure occurs through GitHub Actions; you can see the progress on the GitHub repo under the `Actions` tab, located here. If any example fails, this information is passed on to GitHub and a red X will be shown next to the commit. Otherwise, if all tests pass, a green check mark appears to signify the code changes are valid.

The examples that are covered are shown in `ROSCO/rosco/test/test_examples.py`. If you add an example to ROSCO, make sure to add a call to it in the `run_examples.py` script as well.

## 8.4 Pull requests

Once you have added or modified code, submit a pull request via the GitHub interface. This will automatically go through all of the tests in the repo to make sure everything is functioning properly. The main developers of ROSCO will then merge in the request or provide feedback on how to improve the contribution.

## 8.5 Updating the ROSCO API (Changing Input Files)

Any API changes should result in the following changes:

1. Update to the rosco schema.
2. Update to DISCON writer. You can use the *input_descriptions* dictionary to streamline the process.
3. Document API changes here
4. Update to the rosco registry, which regenerates the ROSCO_IO using this script.

CHAPTER

NINE

# 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:

## 9.1 Bladed versions 4.6 to current (4.12)

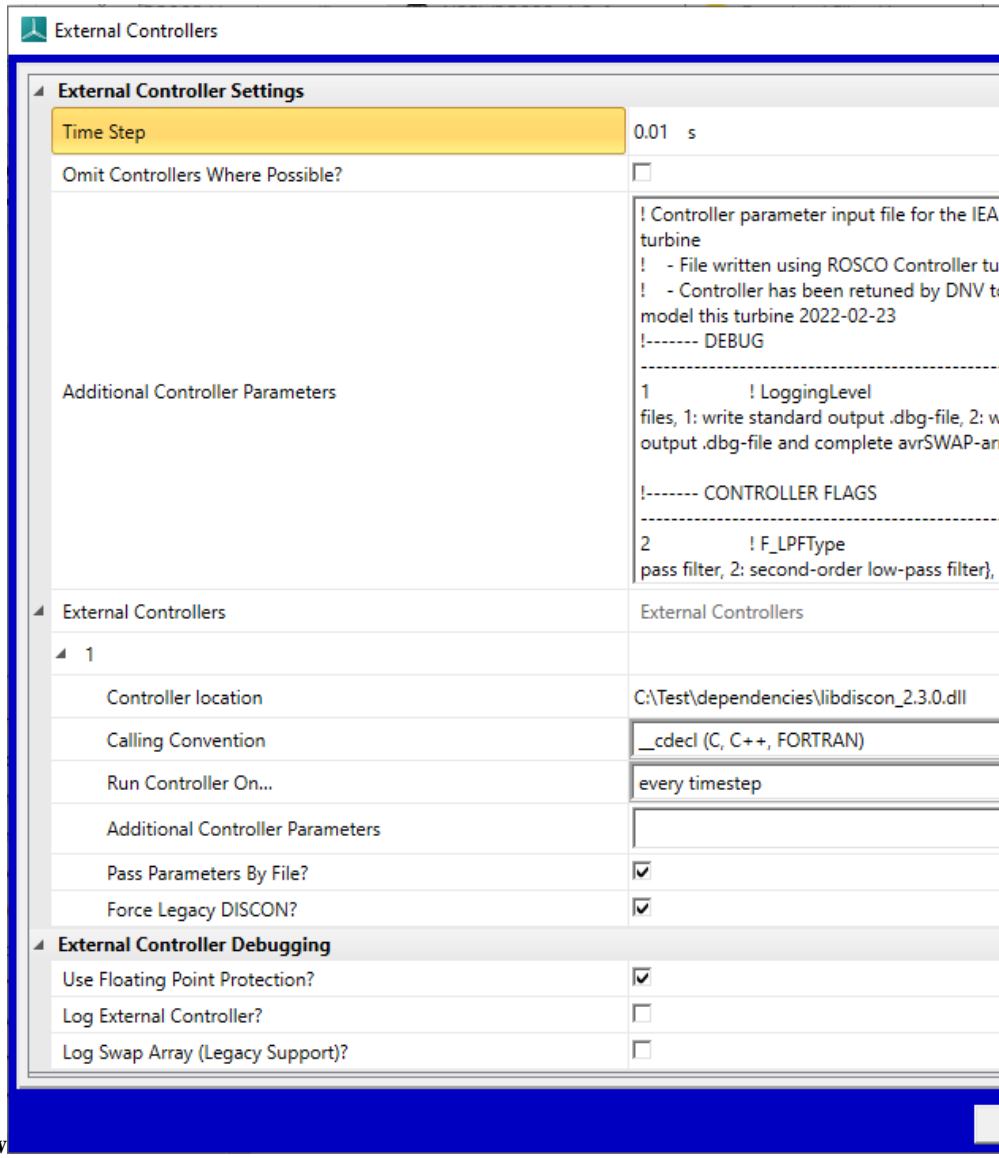
In the Bladed External Controller dialog, fill in the fields as follows:

- *'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**




## 9.2 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

CHAPTER

TEN

# LICENSE

# PYTHON MODULE INDEX

# INDEX

## Symbols



## M