ROSCO

Release 2.10.4

Nikhar J. Abbas, Daniel S. Zalkind

Feb 04, 2026

CONTENTS

Version
2.10.4

Date
Feb 04, 2026

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 gfortran gcc libpython m2-pkg-config  # 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 environment.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 uninstall gcc gfortran  # if you previously installed via pip above
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
- `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.

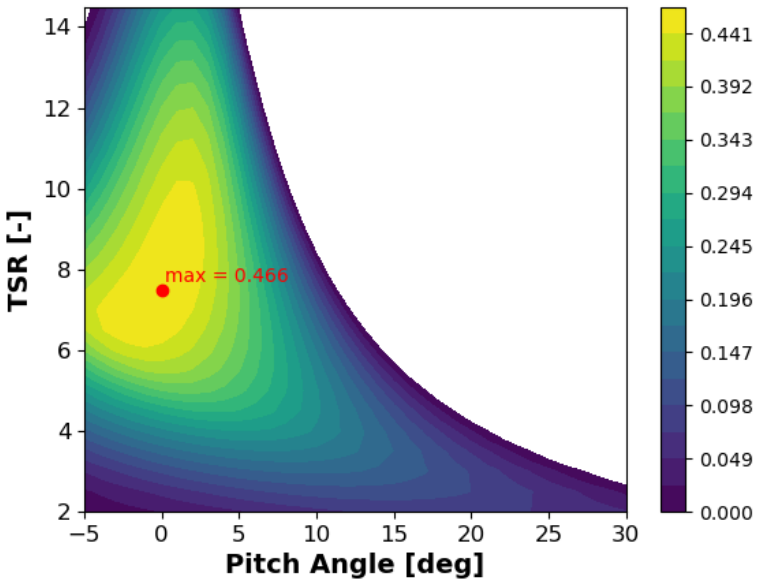
3.4 List of Examples

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

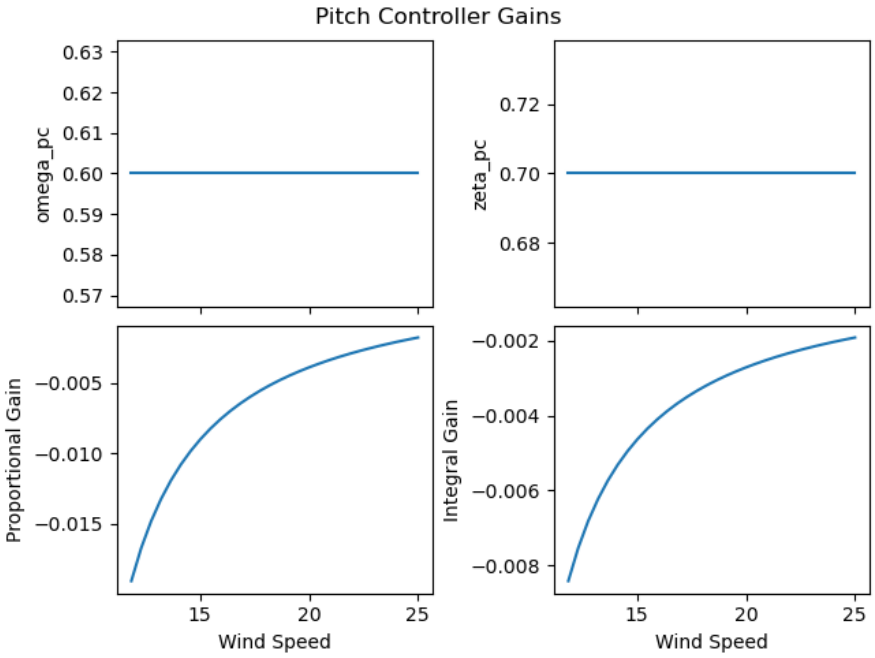
Run CCblade, save a rotor performance text file. In this example:

- 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

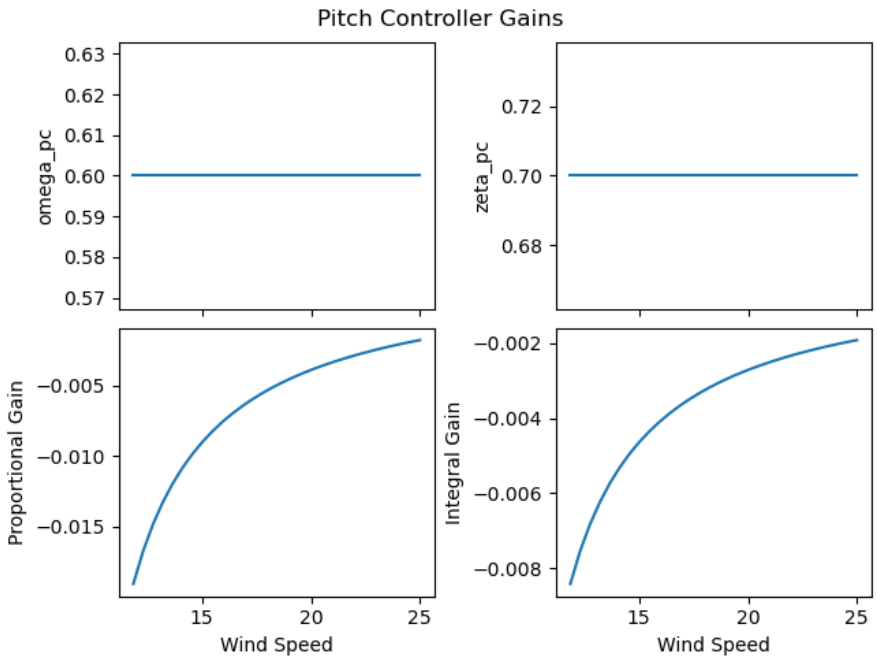
Demonstrate the simple 1-DOF wind turbine simulator with ROSCO

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/

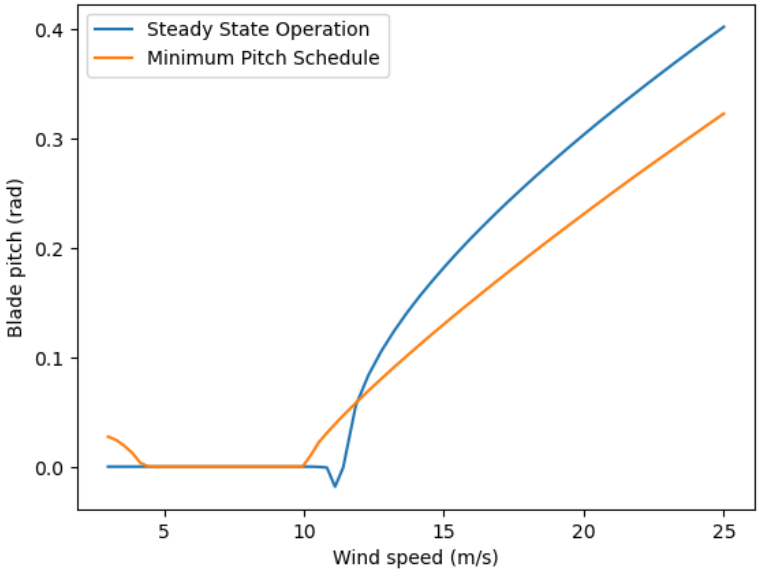
3.4.6 06_peak_shaving

This example demonstrates the minimum pitch schedule of the NREL 5MW RWT. In the ROSCO controller, the minimum pitch is determined from a lookup table from the (estimated) wind speed to the limit. The ROSCO DISCON controller only has PS_Mode of 0 (disable) or 1 (enabled).

When creating a DISCON using the ROSCO toolbox, a few more options are available. For PS_Mode = 1, the minimum pitch limit is calculated to limit the maximum thrust to a fraction of the un-limited maximum thrust, computed from the rotor's Ct table. The `ps_percent` input (to the tuning yaml) is the fraction of the allowed thrust to the maximum, un-limited thrust. For example, if the maximum thrust is 1e5 N, and we want to limite the maximum thrust to 8e4 N, `ps_percent` = 0.8. Note that "percent" is a bit of a misnomer.

For PS_Mode = 2, the minimum pitch limit is calculated to maximize the Cp surface in below rated; this usually results in a non-zero pitch at low wind speeds. For large rotors, sometimes a minimum rotor speed is enforced to avoid resonance between the rotor period and the tower natural frequency; this results in a non-constant TSR across the below-rated operating points. In this case, different pitch angles will be required to maximize Cp.

For PS_Mode = 3, both the thrust limiting and the power maximizing tuning routines determine the minimum pitch limit. This example demonstrates PS_Mode = 3 as follows, compared to the steady state blade pitch operating points of the turbine. The ROSCO toolbox uses Cp and Ct tables to compute these inputs to the ROSCO controller.



3.4.7 07_openfast_outputs

Demonstrate the ROSCO routines for reading OpenFAST I/O files. Nearly all the figures in the example documentation rely on a version of these scripts, which can also be used to load ROSCO *.dbg* files, including

> # .dbg files output basic ROSCO internal variables, like the inputs and outputs to the wind speed estimator
> # .dbg2 files output all the ROSCO LocalVariables. If the variable is an array, only the first entry is recorded. # .dbg3 files output the avrSWAP array at each timestep that is transfered between ROSCO and the multi-physics solver

Note: this example relies on the previous running of the OpenFAST model in 'Test_Cases/NREL-5MW/' to plot.

3.4.8 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.

3.4.9 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

3.4.10 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

3.4.11 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

3.4.12 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

3.4.13 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

3.4.14 16_external_dll

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

3.4.15 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

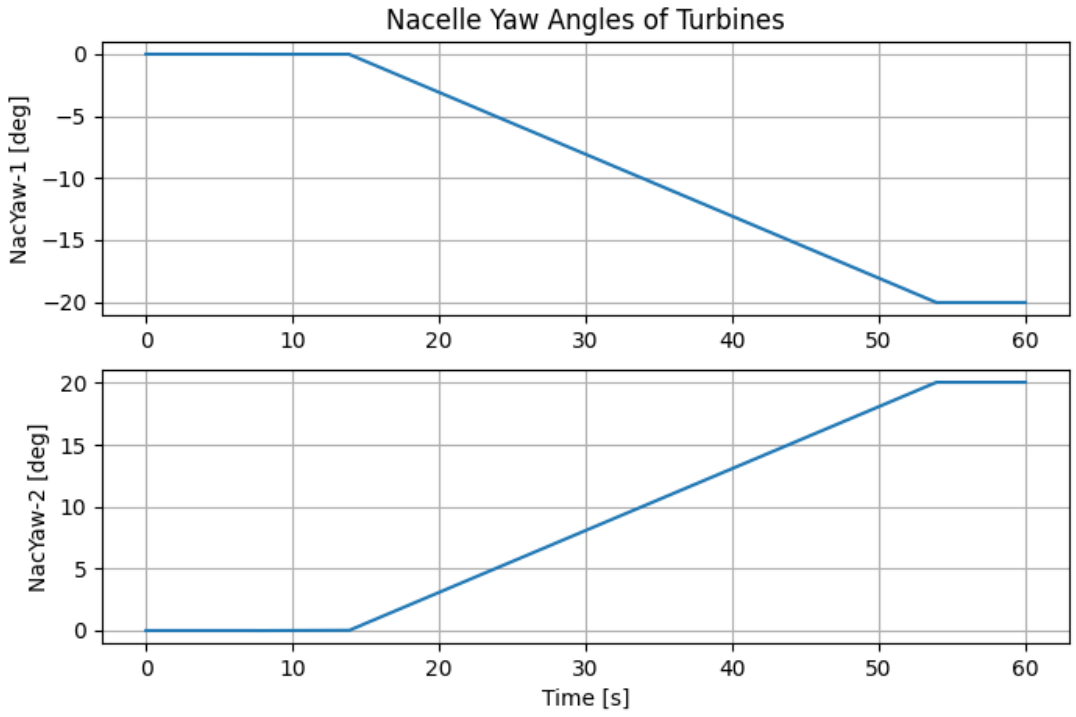
3.4.16 17b_zeromq_multi_openfast

Run multiple openfast simulations and execute communication with ZeroMQ.

3.4.17 17c_zeromq_fastfarm

This example demonstrates the wind farm controls capability of ROSCO using ZeroMQ based communication with a FAST.Farm simulation.

Note that in order to use the wind farm control capability of ROSCO, the `ZMQ_Mode` must be set to 1 and unique identifiers must be set under `ZMQ_ID` for each turbine in the :code. Wind farm level controller should be defined under a class called `wfc_controller` which should implement a method called `update_setpoints`. This method should take as arguments the unique turbine identifier, the current time and measurements, and return the setpoints for the particular turbine. Please see, the file `rosco/controller/rosco_registry/wfc_interface.yaml` for the list of available measurements and setpoints. A ZeroMQ server and FAST.Farm simulation are run in parallel as separate processes. The `wfc_controller` property of the server must be set to the `wfc_controller` class containing the wind farm controller logic. Note that FAST.Farm requires separate ROSCO libraries for each set of OpenFAST files. The nacelle yaw positions outputs of the two turbines in the wind farm are shown below.

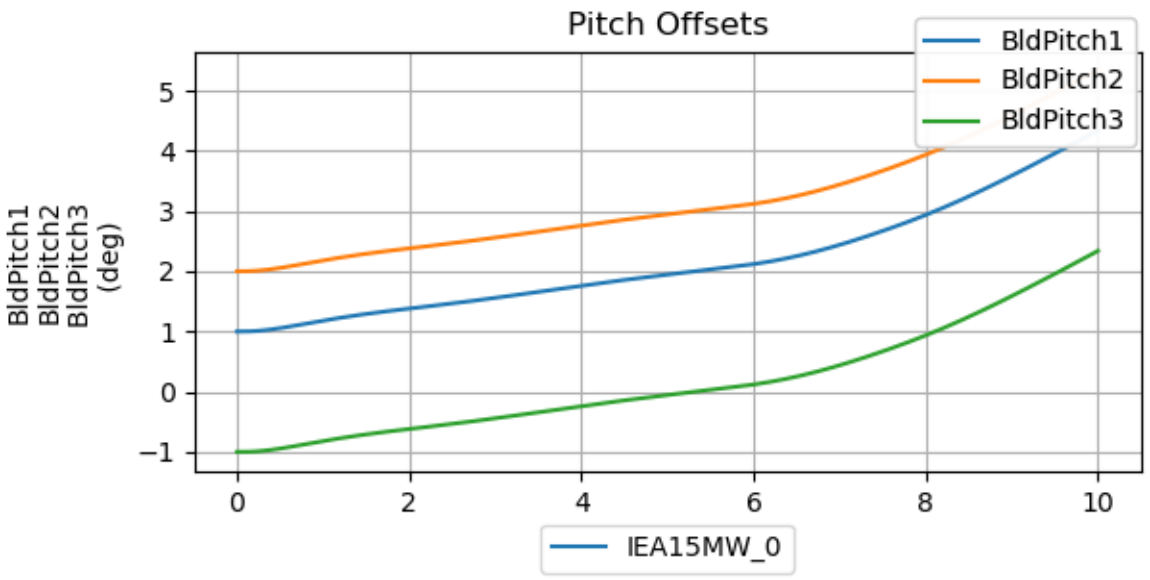


3.4.18 18_pitch_offsets

Demosntrate two kinds of pitch faults using ROSCO:

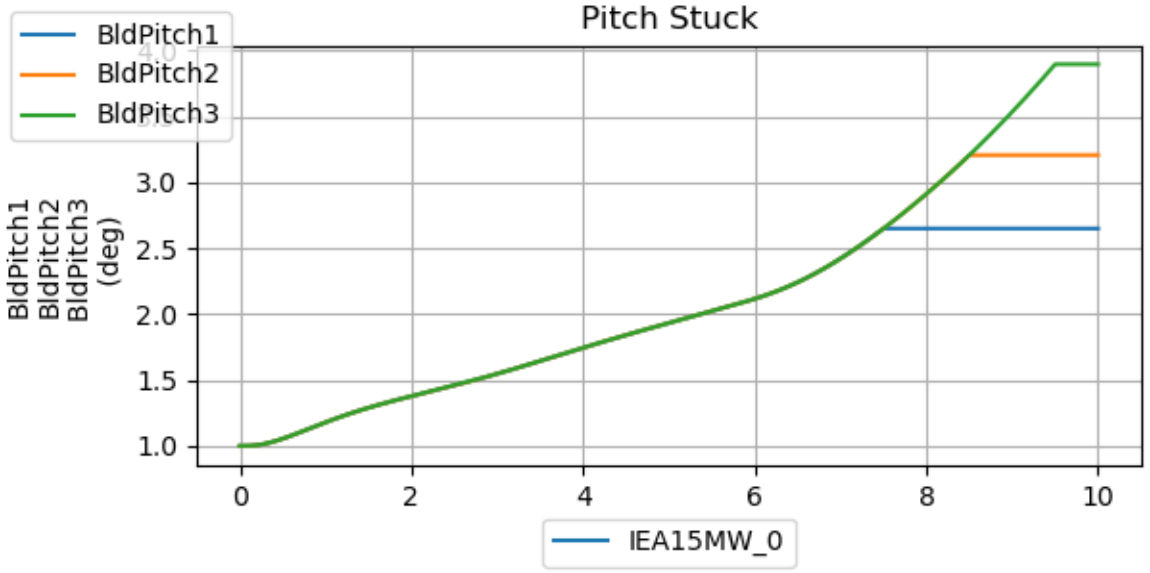
1. Pitch offsets

When `PF_Mode` is set to 1, the pitch controller will apply a constant offset to the pitch angles of the blades. The offsets are set in the `PF_Offsets` array in the DISCON file.



2. Stuck pitch actuator

When `PF_Mode` is set to 2, the pitch actuator will become stuck at its current position at time `PF_TimeStuck`.



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 in the python script.

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_soft_cut_out

Set up a control input to do a soft cut-out of a wind turbine at high wind speeds. This example uses the first power reference control implementation (PRC_Mode of 1), where the user specifies the speed setpoint versus wind speed. We can use this to track specific rotor speeds based on the wind speed, or de-rate/power boost using by changing the speed. With PRC_Mode of 2, we can do this but also change the power rating with pitch and torque.

3.4.28 28_tower_resonance

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

3.4.29 29_power_control

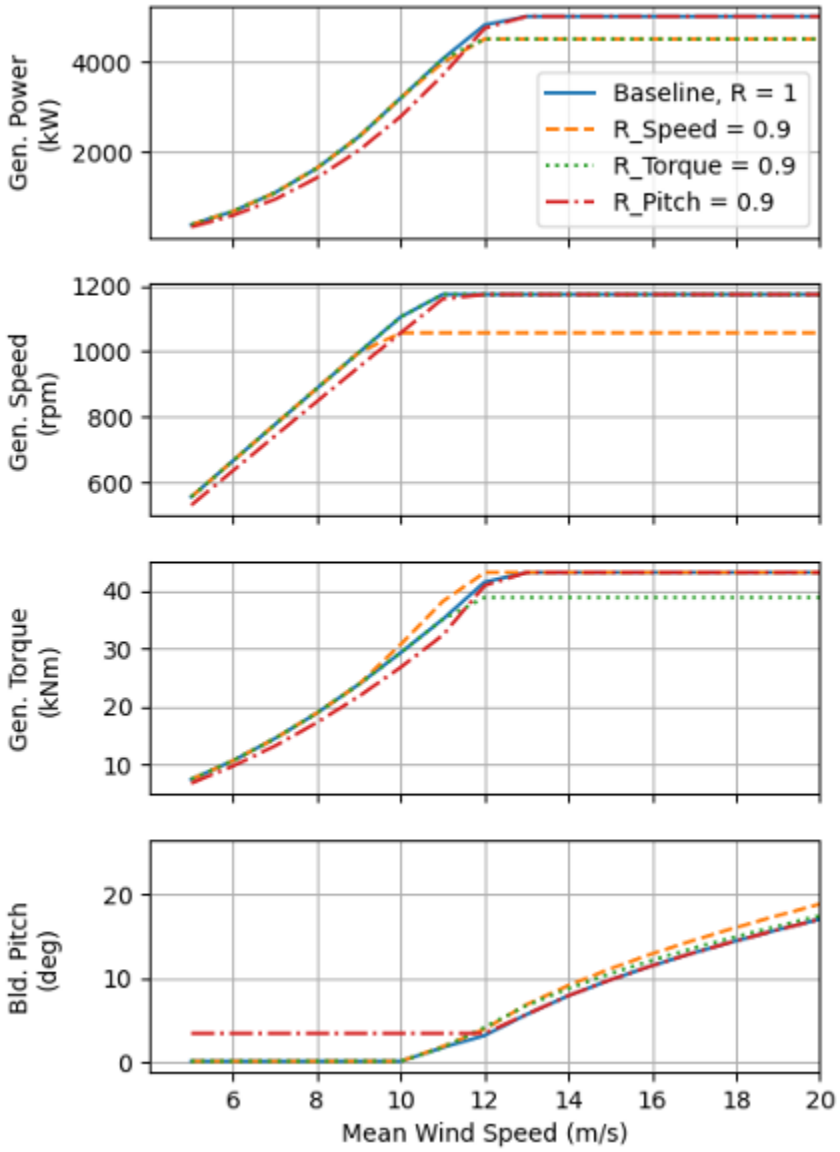
This example demonstrates an advanced method for controlling the power output of a turbine. Users may want to reduce the power output, or de-rate the turbine to reduce loads, increase wind speeds deeper in the farm, or for grid support functions. Users might also want to increase the power output when it's safe to do so. Future advanced control methods can make use of these inputs to directly account for the trade off between power and loads.

There are a few ways to control the turbine power output

The power rating (`R`) is the controlled power relative to the rated power (or available power below rated). For example R = 0.9, will produce 90% of the rated power (or available power below rated).

- Speed (`R_Speed`) will change the rated speed, or the speed setpoint relative to the optimal tip speed ratio (below rated).
- Torque (`R_Torque`): will change the rated torque. In constant power operation (*VS_ConstPower* of 1), the torque is non-constant, but the rated power used to calcualte the torque demand is adapted accordingly.
- Pitch (`R_Pitch`): will change the minimum pitch angle of the turbine. When using peak shaving, the min pitch is the maximum of the minimum pitch for peak shaving (`PS_Min_Pitch`) and the minimum pitch for power control (`PRC_Min_Pitch`). The ROSCO toolbox can generate a lookup table from `R_Pitch` to `PRC_Min_Pitch` using the Cp surface.

The three methods are compared in the following figure:



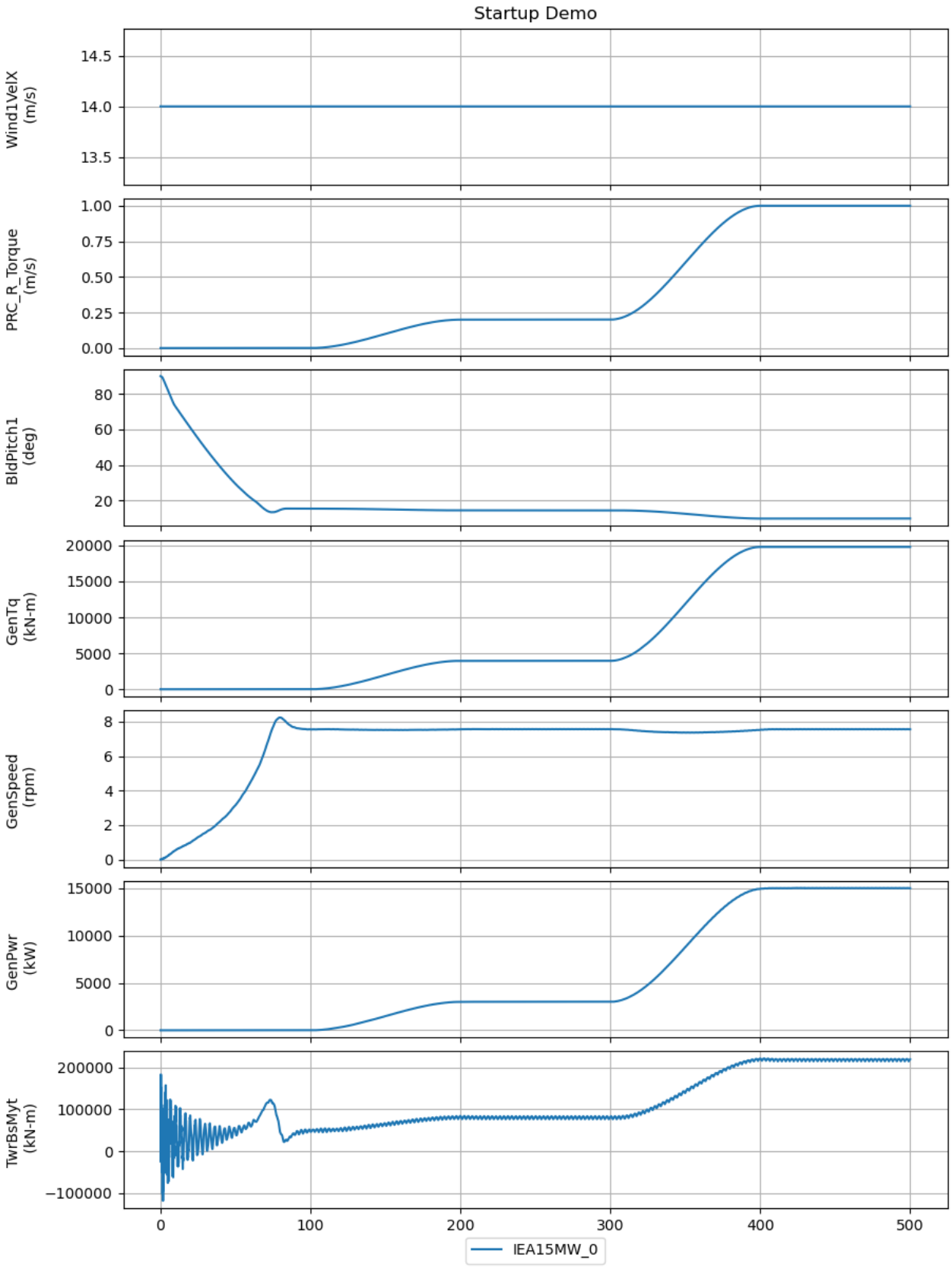
The power rating can be controlled with three different "communication" methods (`PRC_Comm`), via:

1. Constant settings in the DISCON: `R_Speed`, `R_Torque`, and `R_Pitch`.
2. Open-loop control inputs with time or wind speed breakpoints. Two applications will be shown in the example below.
3. The ZeroMQ interface. A simple example is provided in `17b_zeromq_multi_openfast.py`.

This example shows users how to set up open loop control inputs for

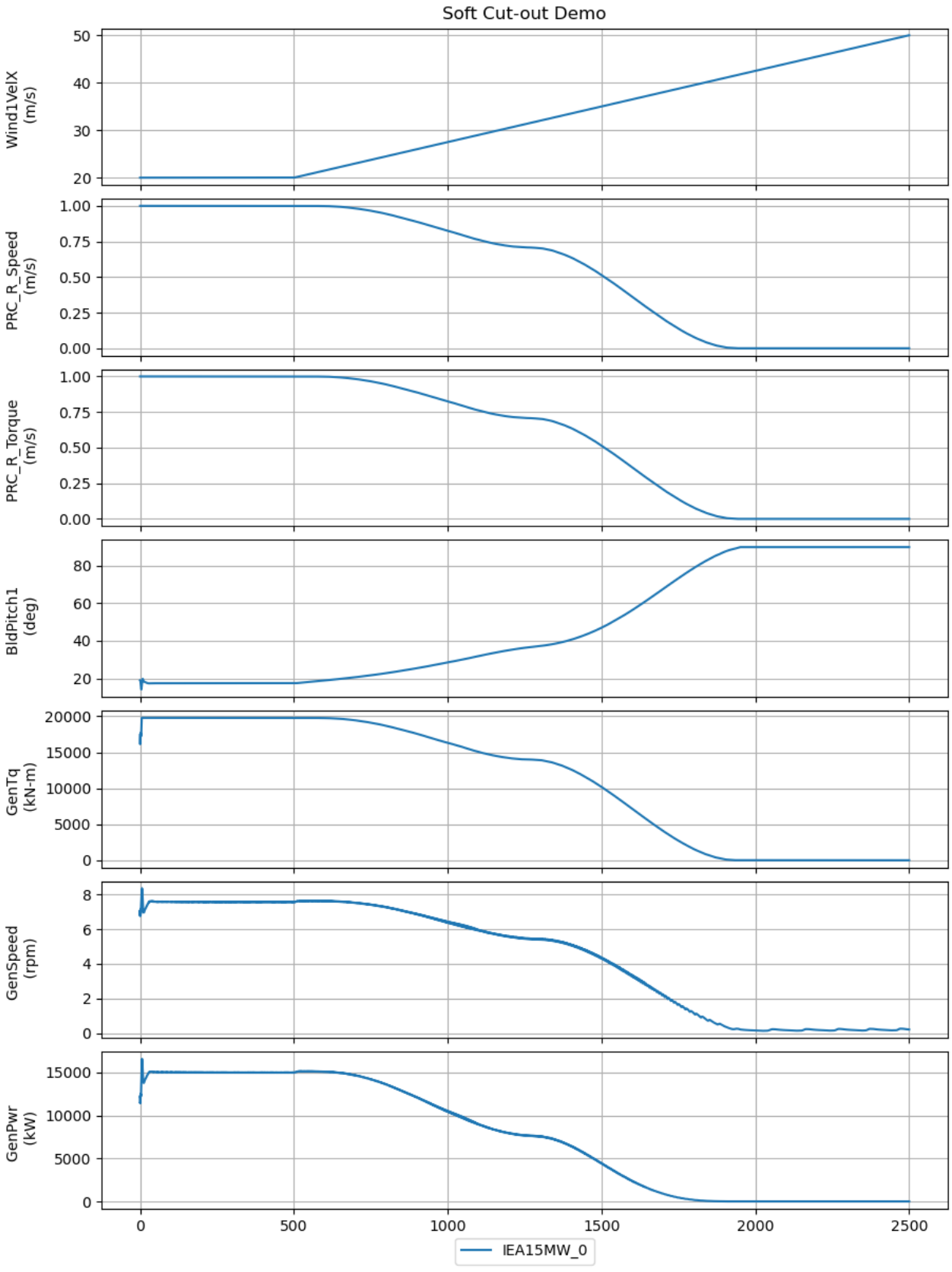
1. A start up routine that ramps the turbine rating using R_Torque in steps over 400 seconds.
2. A soft cut-out routine for high wind speeds.
3. Active wake control for above rated operation, using a sinusoidal input for R_speed.

Start Up Demo



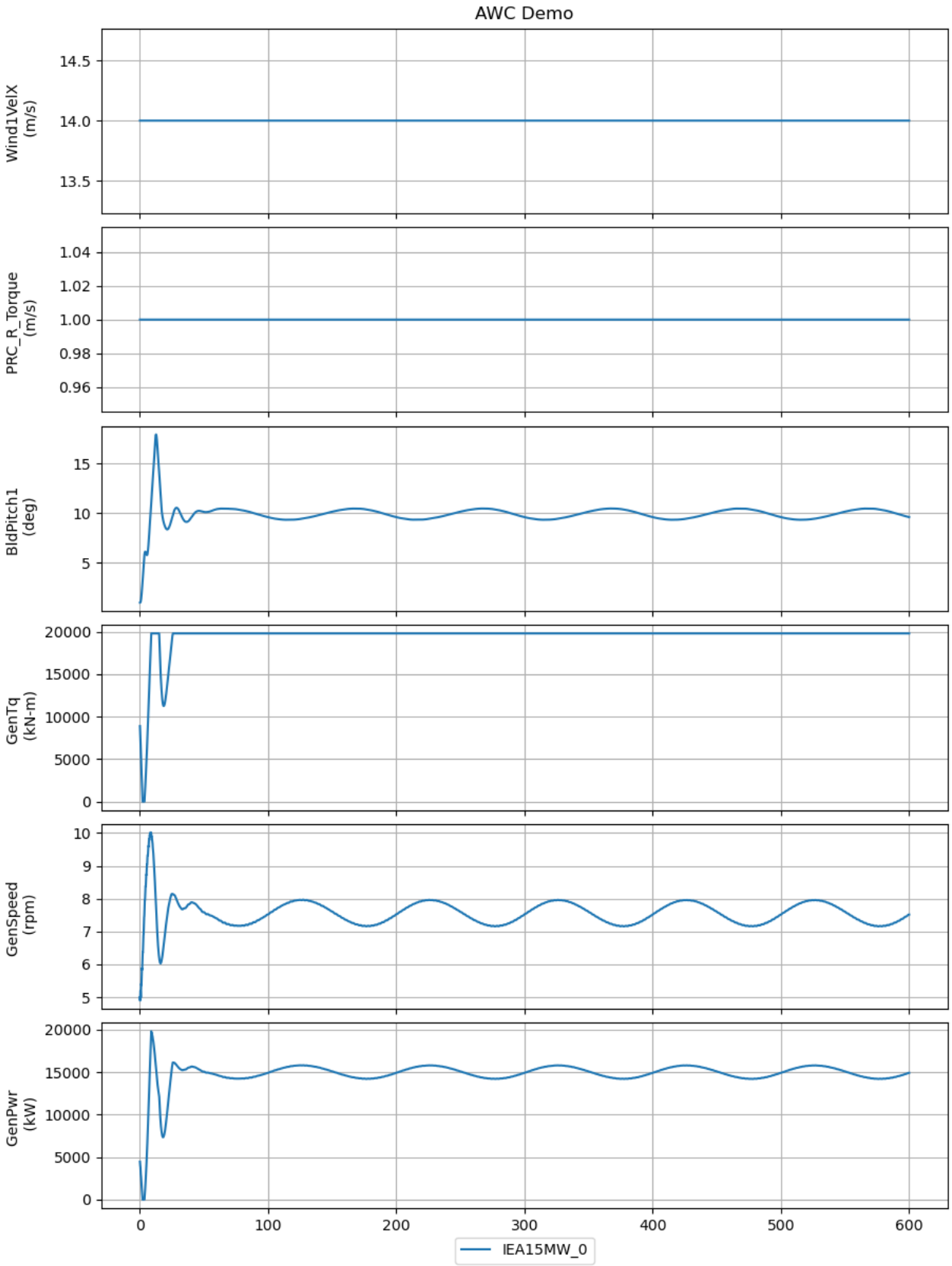
The turbine is started in a parked configuration with the blades pitched to 90 degrees. R_Torque is increased from 0 to 0.2, then from 0.2 to 1.0, simulating a startup routine. The torque, because it is saturated at the max torque, which is scaled by `R_Torque` follows the trajectory of `PRC_R_Torque`. `PRC_R_Torque` is the variable name inside ROSCO and the `.dbg2` file. The blade pitch controller is active throughout the simulation, regulating the generator to the rated speed as usual.

Soft Cut-out Demo



The turbine starts in above-rated operation and a ramp wind input goes well beyond the normal cut-out wind speed of 25 m/s. In this demonstration, we use both R_Speed and R_Torque to ramp the turbine rating from 1.0 at 20 m/s to 0.5 at 30 m/s and 0.0 at 40 m/s. The signals `PRC_R_Speed` and `PRC_R_Torque` ramp towards 0 following the R versus wind speed table. Both GenTq and GenSpeed drop to 0 following the reference signals and the power follows.

Active Wake Control Demo



In active wake control, the goal is to change the rotor thrust through low frequency changes to the blade pitch. Since the blade pitch must be used to control the rotor speed in above rated operation, we can insteady vary the generator speed reference (via R_Speed) to create similar blade pitch variations.

3.4.30 30_shutdown

This example demonstrates turbine shutdown using collective blade pitch threshold mode.

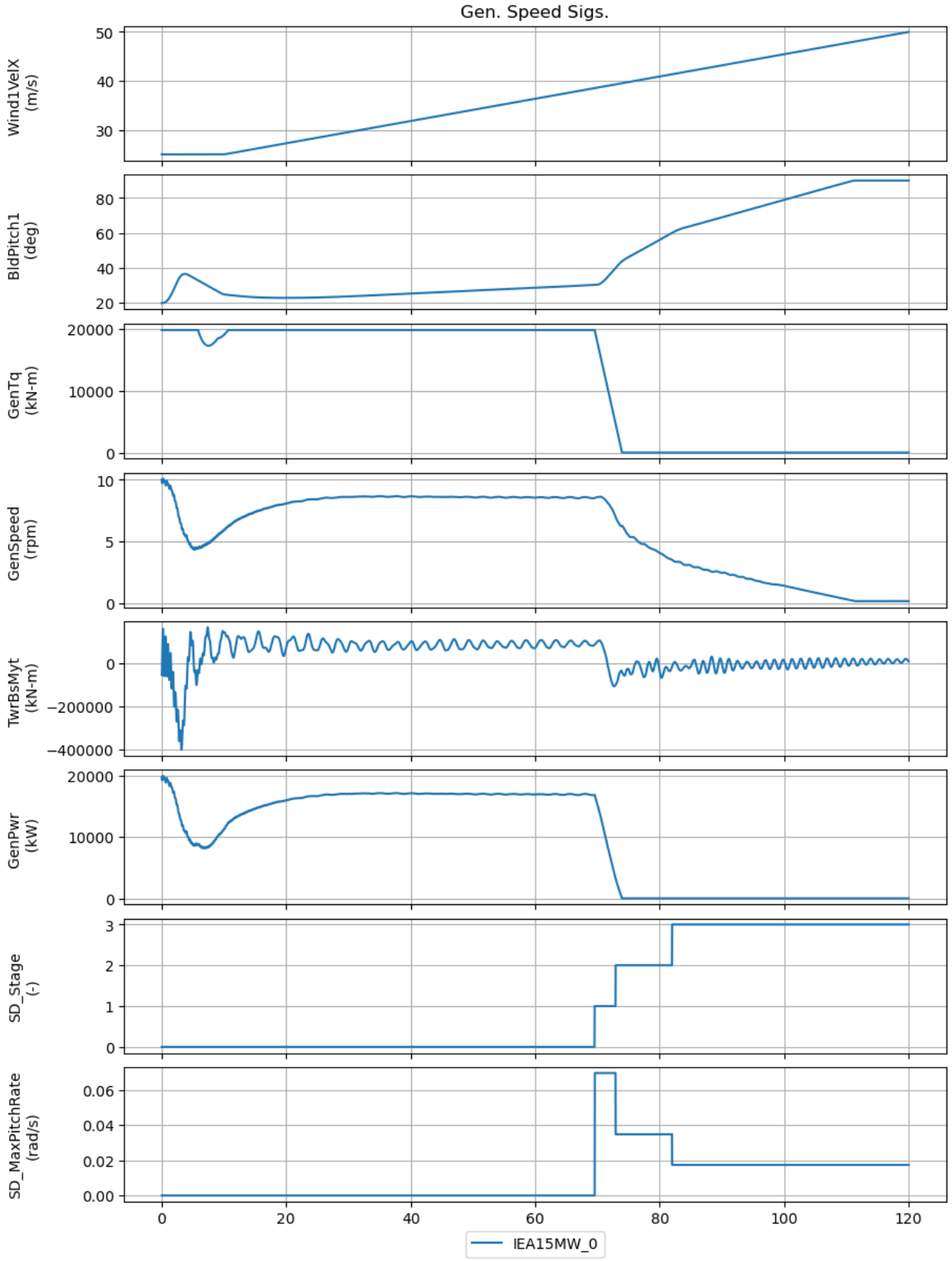
ROSCO allows for four shutdown trigger options:

- collective blade pitch exceeds a threshold
- yaw error exceeds a threshold
- generator speed exceeds a threshold
- Shutdown at a predefined time

There are two methods of shutdown: 1. Timed stages (`SD_Method` = 1): The user specifies a pitch (`SD_MaxPitchRate`) and torque (`SD_MaxTorqueRate`) rate for each stage, and the stage lasts for a specified time (`SD_Stage_Time`). 2. Pitch-based stages (`SD_Method` = 2): The user specifies a pitch and torque rate for each stage, and the stage lasts until the pitch is above the specified thresholds (`SD_Stage_Pitch`).

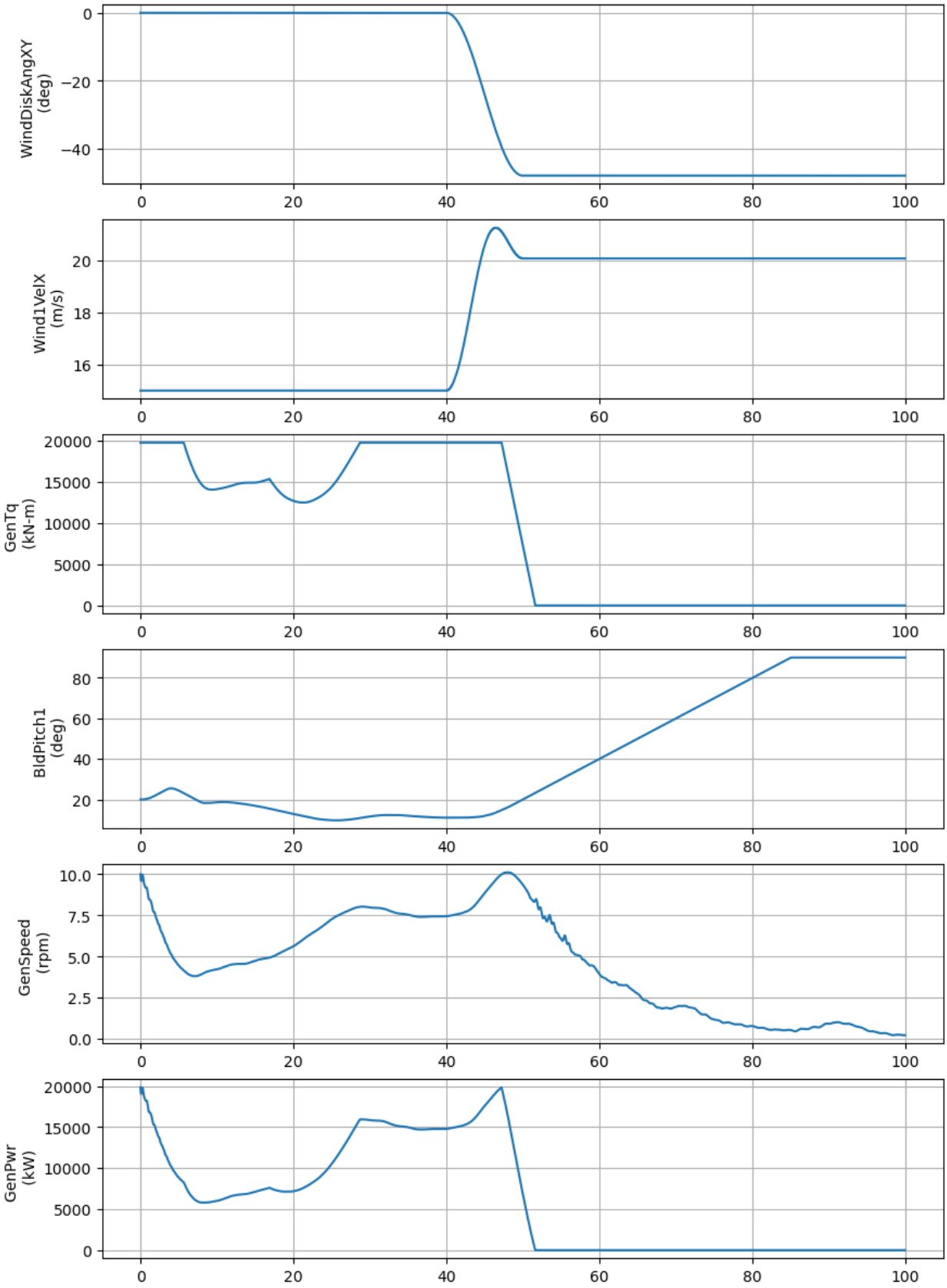
Pitch Threshold Demo

The following plot demonstrates turbine shutdown when blade pitch exceeds a threshold of 30 degrees. This shutdown mode can provide protection against high wind speeds.


Gen. Speed Sigs.
Wind1VelX (m/s)
BldPitch1 (deg)
GenTq (kN-m)
GenSpeed (rpm)
TwrBsMyt (kN-m)
GenPwr (kW)
SD_Stage (-)
SD_MaxPitchRate (rad/s)
IEA15MW_0

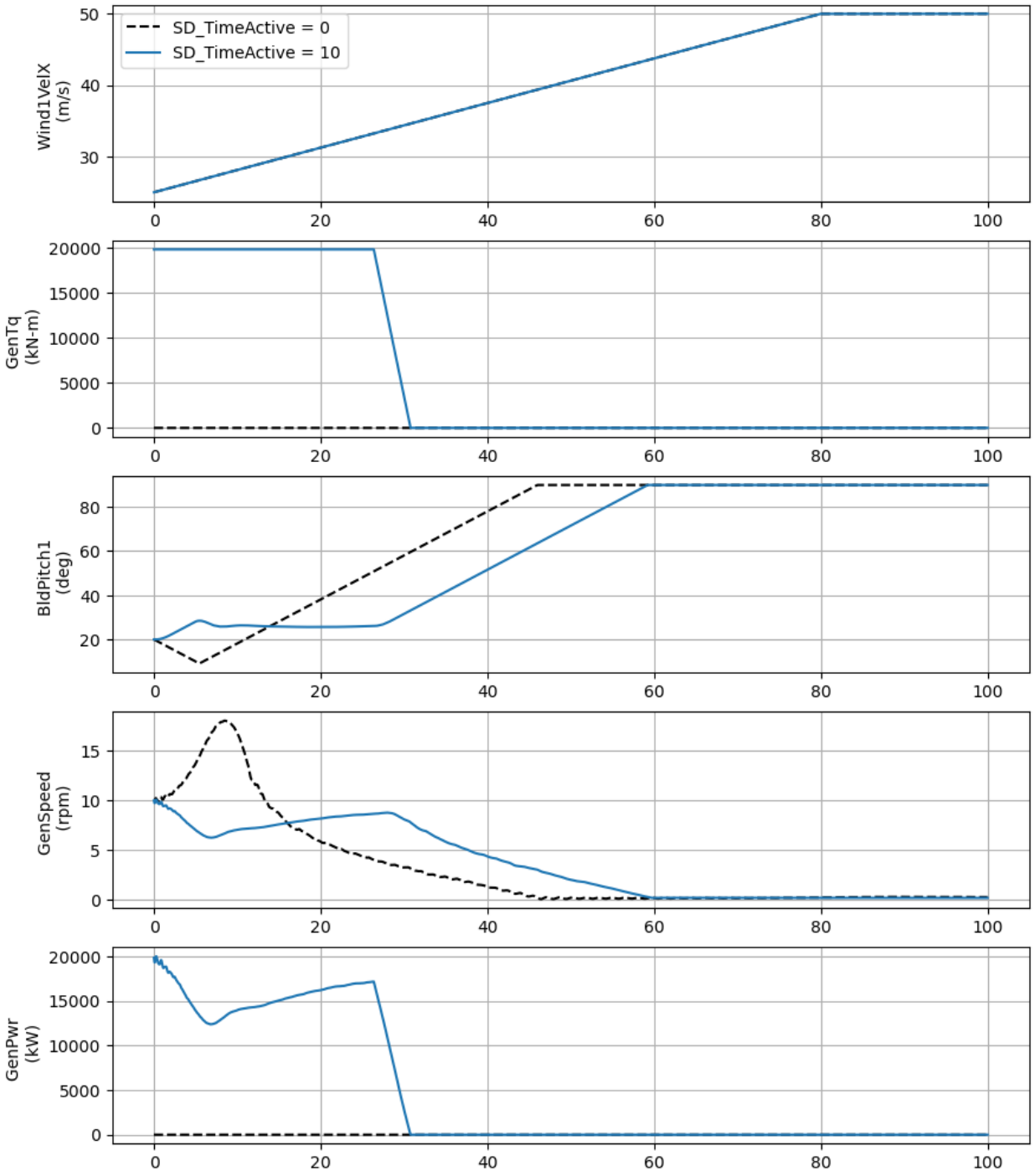
Yaw Error Threshold Demo

The following plot demonstrates turbine shutdown when turbine yaw error pitch exceeds a threshold of 25 degrees. This demonstration uses the extreme coherent gust with direction change wind inflow used in DLC 1.4.


WindDiskAngXY (deg)
0
−20
−40
0
20
40
60
80
100
Wind1VelX (m/s)
20
18
16
GenTq (kN-m)
20000
15000
10000
5000
0
BldPitch1 (deg)
80
60
40
20
GenSpeed (rpm)
10.0
7.5
5.0
2.5
0.0
GenPwr (kW)

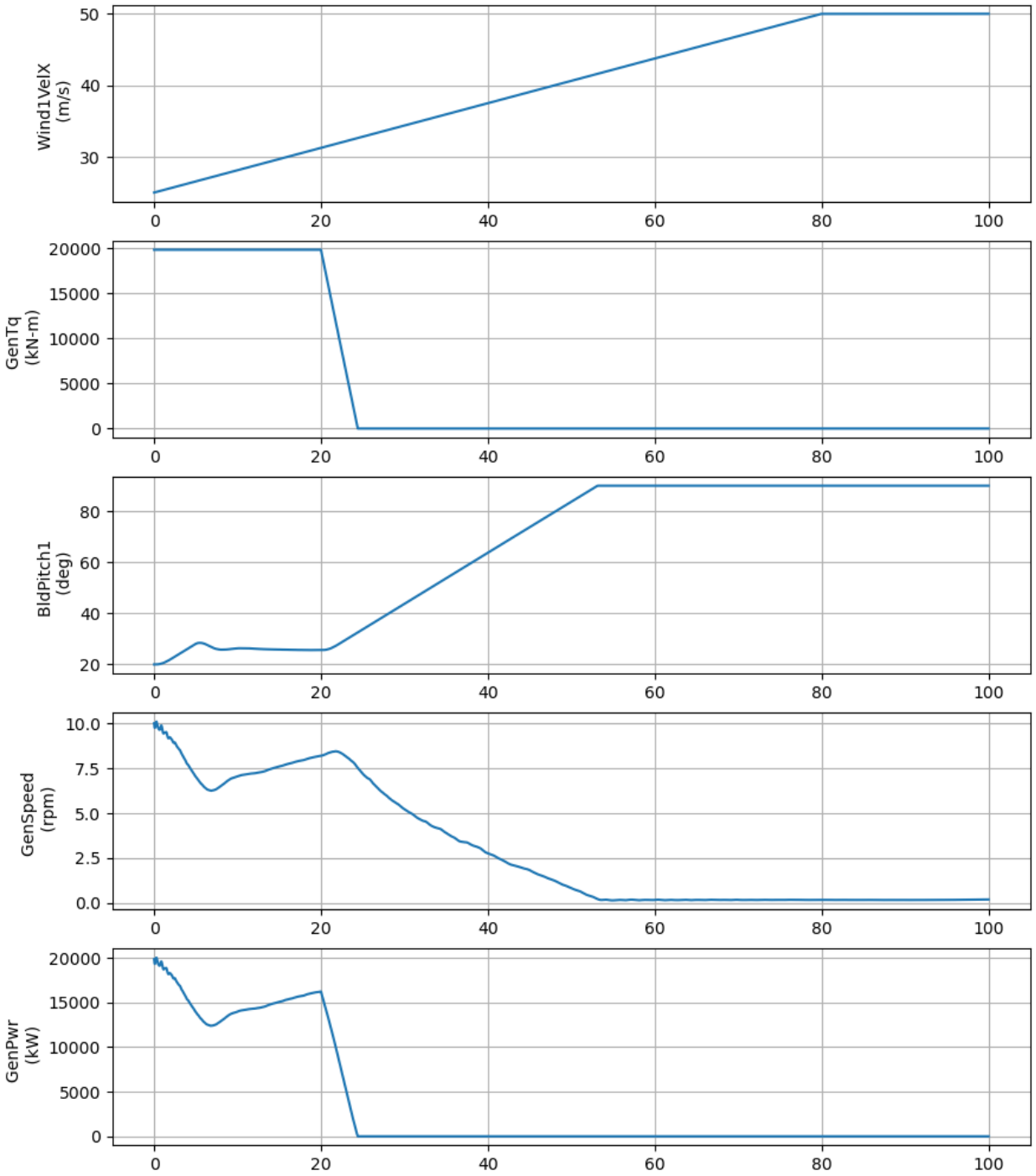
Generator Speed Threshold Demo

The following plot demonstrates turbine shutdown when generator speed exceeds a threshold of 8.5 rpm. This also compares the use of `SD_TimeActive` to enable shutdown at 0 seconds and 10 seconds.



Time Demo

The following plot demonstrates turbine shutdown at 20 second time.



3.4.31 31_fixed_pitch_mhk

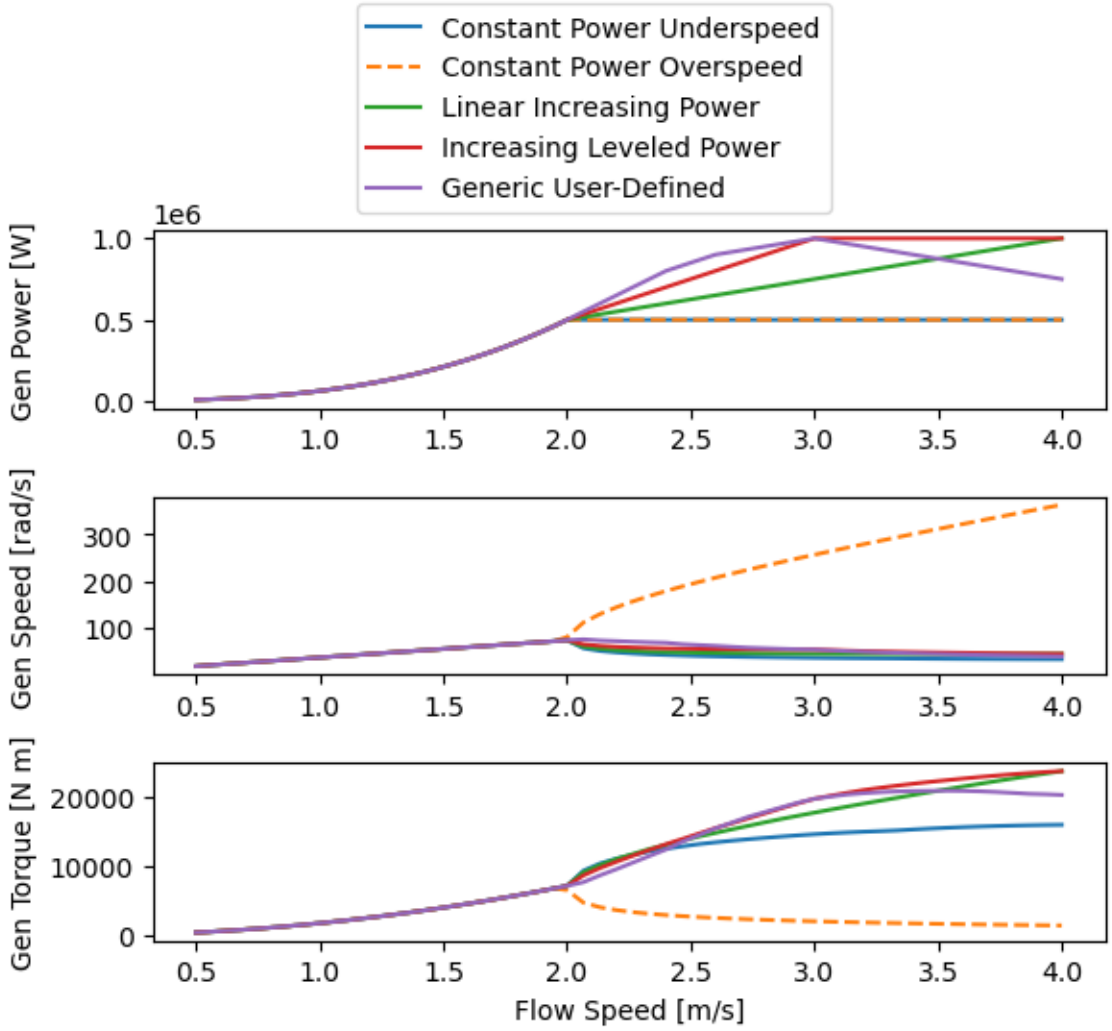
This example demonstrates the fixed-pitch control of a marine hydrodkinetic (MHK) turbine.

There are several ways to control the power output of a turbine in above-rated conditions. In this example we demonstrate the following control configurations:

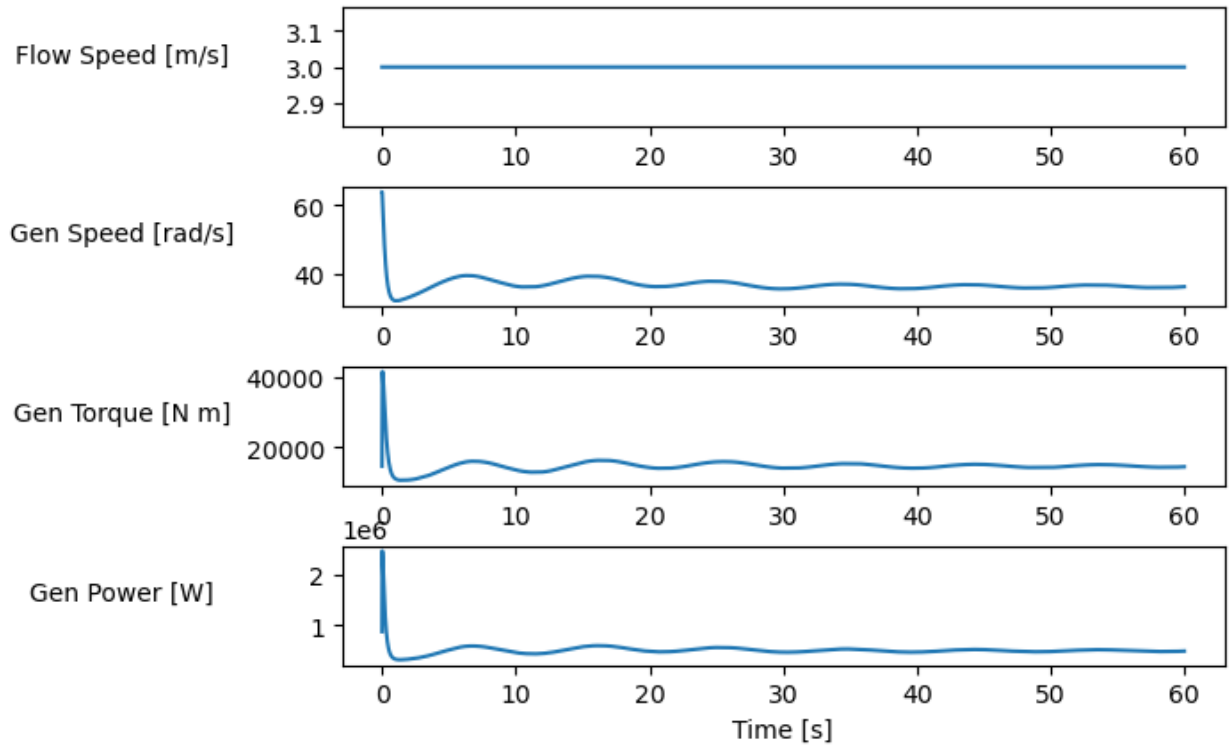
1. Constant power underspeed (should be the default)
2. Constant power overspeed
3. Linear increasing power
4. Linear increasing power, leveling out
5. Generic numeric function
6. Constant power overspeed, nonlinear lookup table control

More details about the controller methods can be found in *ROSCO Control of Marine Hydrokinetic Turbines (MHKs)*.

The desired power curves of each configuration are as follows:



In the first case, the reference generator speed is decreased (underspeed) to maintain a constant rated power above rated. To slow down the generator, a higher torque must be used:



3.4.32 32_Startup

This example demonstrates turbine startup procedure. The startup occurs in various stages depending on the provided input. The states are as follows:

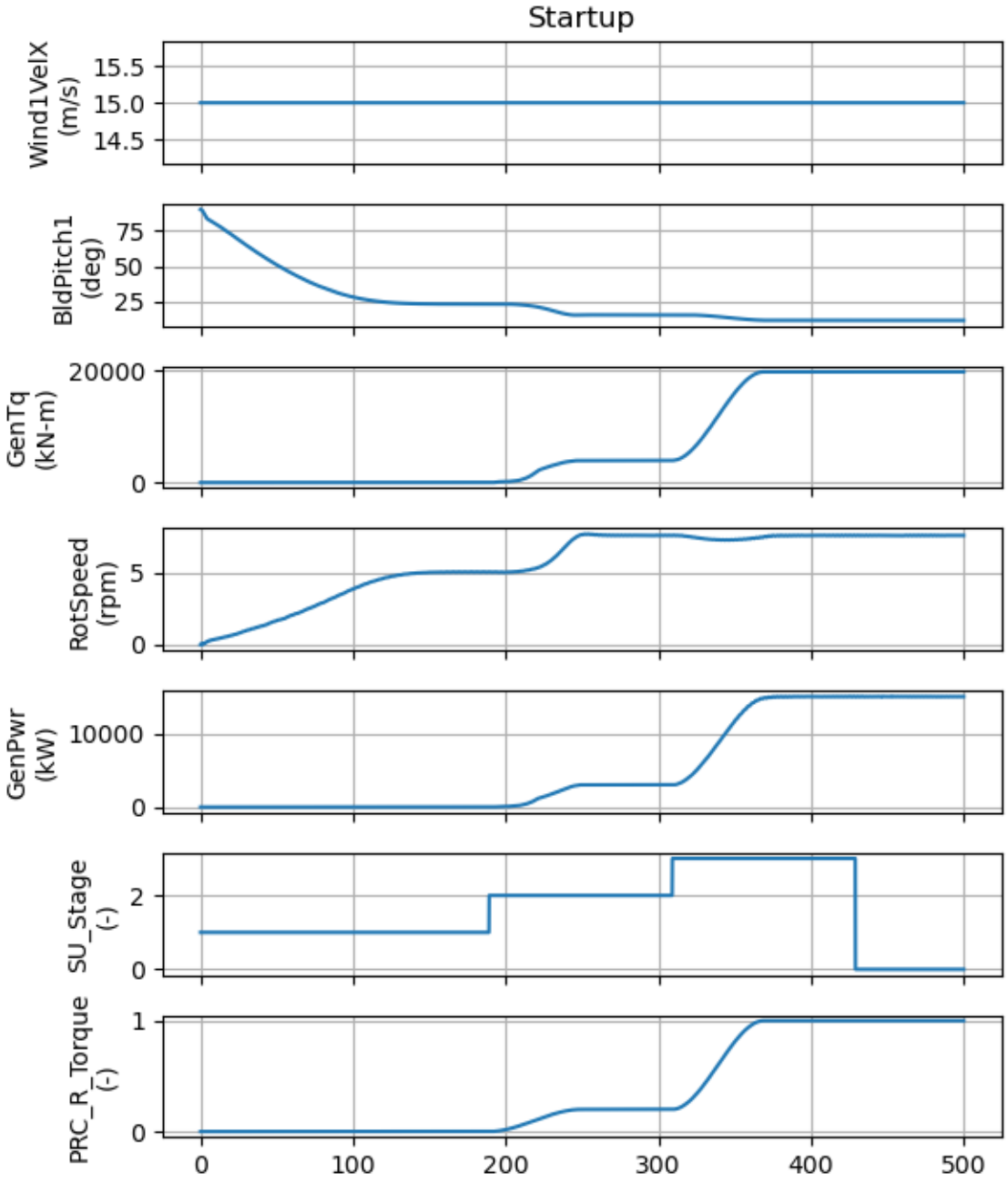
- Stage 1: Free-Wheeling of rotor

  In this stage the generator torque is 0 (set using `PRC_R_Torque = 0`) and the rotor is free to rotate. The rotor continues to free-wheel until the rotor speed exceeds `0.95 * SU_RotorSpeedThresh` and stay above it for at least `SU_FW_MinDuration` seconds. Once both these criteria are met, the startup procedure enters the next stage.

- **Stage 2 - [`SU_LoadStages_N + 2`]:**
  In the next stage(s), the startup procedure relinquishes the control of the blade pitch angles to the normal speed controller. The power and load on the rotor is increased in stages by changing `PRC_R_Torque`. ROSCO reads an array of desired power loads as `SU_LoadStages`. It also reads ramp-duration and hold-duration for each load stage as `SU_LoadRampDuration` and `SU_LoadHoldDuration` respectively. At each stage, the startup procedure increases `PRC_R_Torque` from its previous value to its current value using a smooth ramp in the form of a sigma function. The duration of the ramp is determined by the `SU_LoadRampDuration` array. Then, the `PRC_R_Torque` held constant at the current level for a period of time (`SU_LoadHoldDuration`).

The stage is stored in a ROSCO local variable `SU_LoadStage`. After the startup procedure is complete, `SU_LoadStage` resets to 0.

The following figure demonstrate the startup procedure

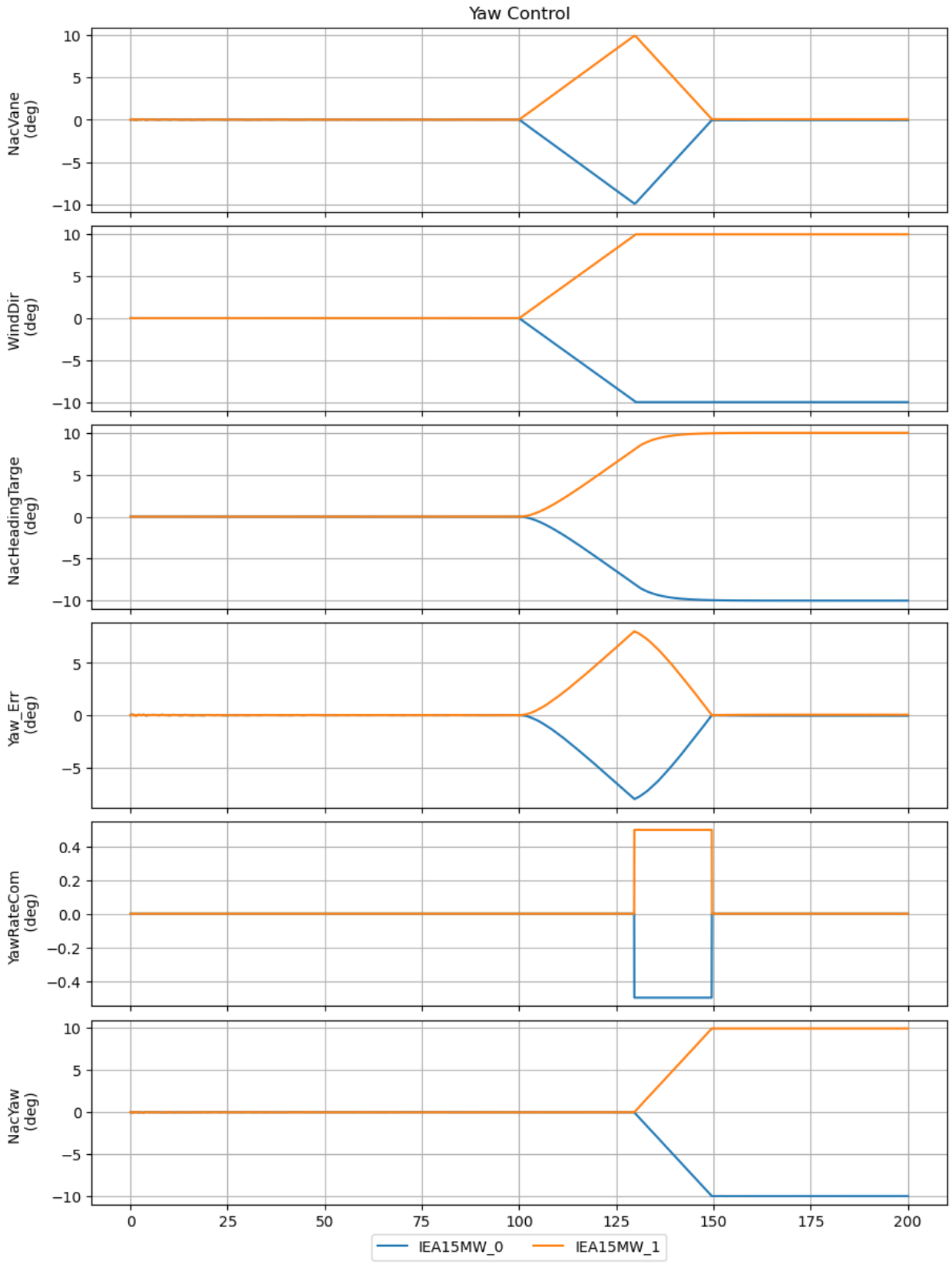


3.4.33 33_yaw_control

This example demonstrates how to run the ROSCO yaw controller. Details of the controller can be found and referenced here: https://wes.copernicus.org/articles/5/451/2020/

The nacelle wind vane signal (`NacVane`) is filtered and, along with offsets either from `Y_MErrSet` or `ZMQ_YawOffset`, a target (`NacHeadingTarget`) is defined. The error (`Yaw_Err`) is the difference between the target and the current nacelle heading. The wind speed dependent deadband (`Y_ErrThresh`) determines when the yaw controller will activate. A state machine will yaw the turbine at a constant rate (`Y_Rate`) until the target is reached.

The following time series should be generated by the example:


Yaw Control
NacVane (deg)
WindDir (deg)
NacHeadingTarge (deg)
Yaw_Err (deg)
YawRateCom (deg)
NacYaw (deg)
IEA15MW_0
IEA15MW_1

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.

Detailed and up-to-date documentation of the DISCON inputs can be found in the *ROSCO_Toolbox tuning .yaml* page in controller_params -> DISCON.

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*.

A detailed account of the inputs is automatically generated from the rosco toolbox_schema.yaml file, which is located in the `rosco/toolbox/inputs` folder:

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 2.9.0 to 2.10.0

New power reference control mode

- De-rate or power boost the turbine using the reference speed, rated torque, or minimum pitch. More details can be found in the Examples page.

New startup mode

- Startup the turbine by allowing the rotor to rotate freely (free-wheel) and the load rotor in stages to start the turbine.

Updated shutdown mode

- Shutdown mode is modified to enable shutdown based on a set of triggers. The triggers include shutdown on exceeding a predefined pitch, yaw error or generator speed threshold; or at a specific time.

Fixed blade pitch control mode

- The fixed blade pitch mode will use torque control only for the generator and stall-regulated control.

Removed in ROSCO 2.10.0

Line	Input Name	Example Value
129	SD_MaxPit	0.4363 ! SD_MaxPit - Maximum blade pitch angle to initiate shutdown, [rad]
130	SD_CornerFre	0.4188 ! SD_CornerFreq - Cutoff Frequency for first order low-pass filter for blade pitch angle, [rad/s]

Changed in ROSCO 2.10.0

Line	Input Name	Example Value
19	PRC_M	0 ! PRC_Mode - Power reference tracking mode{0: power control disabled, 1: lookup table from wind speed to generator speed setpoints, 2: change speed, torque, pitch to control power}

New in ROSCO 2.10.0 Line	Input Name	Example Value
15	VS_FBP	0 ! VS_FBP - Fixed blade pitch configuration mode (0-
22	SU_Mode	0 ! Startup mode {0: no startup procedure, 1: startup en
54	F_VSRefSpdCornerFreq	0.20944 ! F_VSRefSpdCornerFreq - Corner frequency (
96	Fixed Pitch Section	!------- FIXED PITCH REGION 3 TORQUE CONTRO
97	VS_FBP_n	60 ! VS_FBP_n - Number of gain-scheduling table entr
98	VS_FBP_U	3.000000 3.266897 3.533793 3.800690 4.067586 4.334
99	VS_FBP_Omega	0.523599 0.523599 0.523599 0.523599 0.523599 0.523
100	VS_FBP_Tau	681375.170448 879890.618036 1113642.754997 13855
107	PRC_Comm	0 ! PRC_Comm - Power reference communication mod
108	PRC_R_Torque	1.00000 ! PRC_R_Torque - Constant power rating throu
109	PRC_R_Speed	1.00000 ! PRC_R_Speed - Constant power rating throug
110	PRC_R_Pitch	1.00000 ! PRC_R_Pitch - Constant power rating throug
111	PRC_Table_n	20 ! PRC_Table_n - Number of elements in PRC_R to _
112	PRC_R_Table	0.0000 0.0526 0.1053 0.1579 0.2105 0.2632 0.3158 0.3
113	PRC_Pitch_Table	0.2296 0.2222 0.2144 0.2066 0.1984 0.1902 0.1814 0.1
155	Start Up Section	!------- STARTUP ----------
156	SU_StartTime	120.0000000000 ! SU_StartTime - Time to start startup
157	SU_FW_MinDuration	200.000 ! SU_FW_MinDuration - Free-wheel minimun
158	SU_RotorSpeedThresh	0.5200 ! SU_RotorSpeedThresh - Rotor speed threshhol
159	SU_RotorSpeedCornerFreq	0.4188 ! SU_RotorSpeedCornerFreq - Cutoff Frequency
160	SU_LoadStages_N	2 ! SU_LoadStages_N - Number of load staged for start
161	SU_LoadStages	0.2000 1.0000 ! SU_LoadStages - Array containing loa
162	SU_LoadRampDuration	100.0000 100.0000 ! SU_LoadRampDuration - Array c
164	SU_LoadHoldDuration	200.0000 100.0000 ! SU_LoadHoldDuration - Array co
165	Shutdown Section	!------- SHUTDOWN ----------
166	SD_TimeActivate	0 ! SD_TimeActivate - Time to acitvate shutdown mode
167	SD_EnablePitch	0 ! SD_EnablePitch - Shutdown when collective blade p
168	SD_EnableYawError	0 ! SD_EnableYawError - Shutdown when yaw error ex
169	SD_EnableGenSpeed	0 ! SD_EnableGenSpeed - Shutdown when generator sp
170	SD_EnableTime	0 ! SD_EnableTime - Shutdown at a predefined time, [-
171	SD_MaxPit	0.393860000000 ! SD_MaxPit - Maximum blade pitch
172	SD_PitchCornerFreq	0.418880000000 ! SD_PitchCornerFreq - Cutoff Freque
173	SD_MaxYawError	30.00000000000 ! SD_MaxYawError - Maximum yaw
174	SD_YawErrorCornerFreq	0.418880000000 ! SD_YawErrorCornerFreq - Cutoff Fr
175	SD_MaxGenSpd	0.950020000000 ! SD_MaxGenSpd - Maximum genera
176	SD_GenSpdCornerFreq	0.418880000000 ! SD_GenSpdCornerFreq - Cutoff Fre
177	SD_Time	9999.000000000 ! SD_Time - Shutdown time, [s]
178	SD_Method	1 ! SD_Method - Shutdown method {1- Reduce generat
179	SD_Stage_N	1 ! SD_Stage_N - Number of shutdown stages (should e
180	SD_StageTime	1000.0000 ! SD_StageTime - Array containing the time
181	SD_StagePitch	1.5708 ! SD_StagePitch - Array with pitch angles to rea
182	SD_MaxTorqueRate	225000.0000 ! SD_MaxTorqueRate - Maximum torque
183	SD_MaxPitchRate	0.0044 ! SD_MaxPitchRate - Maximum pitch rate used
198	OL_BP_Mode	0 ! OL_BP_Mode - Breakpoint mode for open loop con
199	OL_BP_FiltFreq	0.000000 ! OL_BP_FiltFreq - Natural frequency of 1st
208	Ind_R_Speed	0 ! Ind_R_Speed - Index (column, 1-indexed) of power
209	Ind_R_Torque	0 ! Ind_R_Torque - Index (column, 1-indexed) of power
210	Ind_R_Pitch	0 ! Ind_R_Pitch - Index (column, 1-indexed) of power r

6.2 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 acceller-ation 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 2.9.0

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 de-nominator, resp) - determines the width and depth of the notch, [-]

New in ROSCO 2.9.0 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_NotchBetaI	0.0000 ! F_NotchBetaNum - Damping value of numerator (determines the width of notch). Array with length F_NumNotchFilts, [-]
41	F_NotchBetaI	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_ExclSpe	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]

6.3 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_Groupl	2818 2838 2858 ! StC_GroupIndex - First index for structural control group, options specified in ServoDyn summary output

6.4 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	

6.5 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]

6.6 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_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

CHAPTER

SEVEN

ROSCO_TOOLBOX TUNING .YAML

Definition of inputs for ROSCO tuning procedure

The inputs to the rosco tuning yaml are defined in the rosco toolbox_schema.yaml file, which is located in the `rosco/toolbox/inputs` folder. The following is automatically generated from that file:

CHAPTER

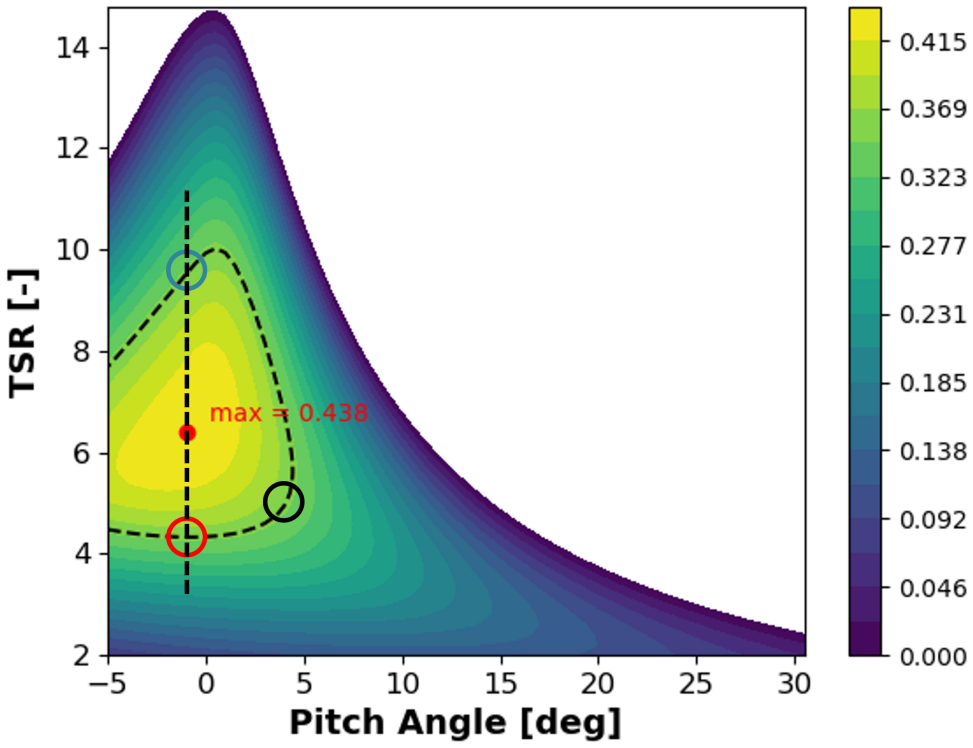
EIGHT

ROSCO CONTROL OF MARINE HYDROKINETIC TURBINES (MHKS)

8.1 Introduction

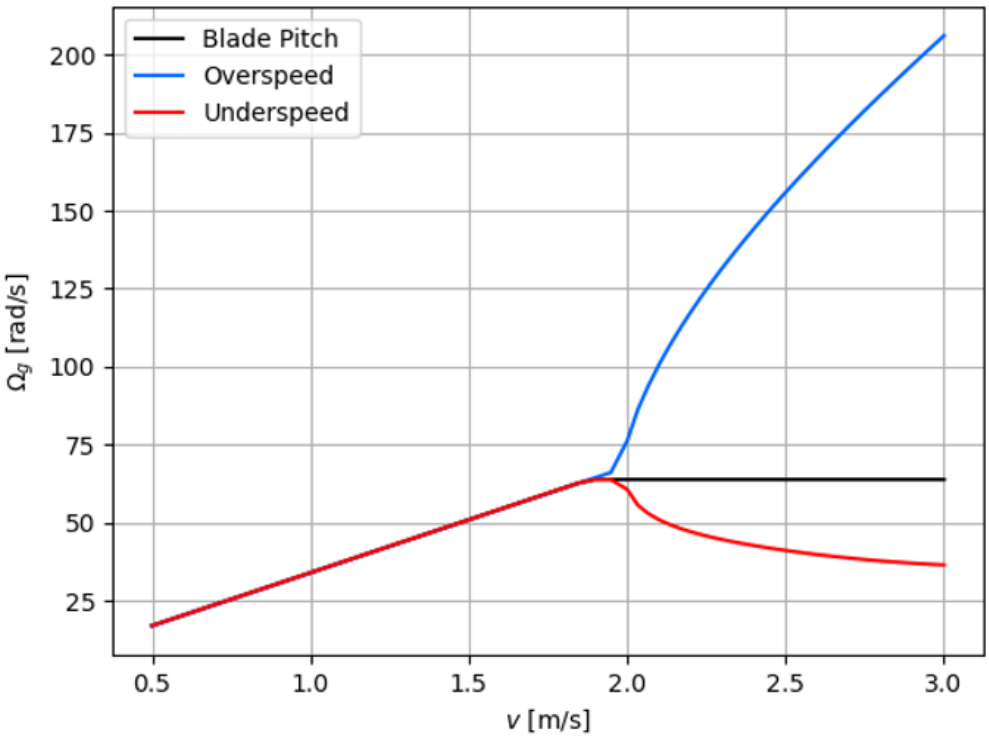
Here, we detail the control of MHK turbines in above rated flow speeds. In below rated flow speeds, torque control is used to operate the turbine at it's maximum tip speed ratio (TSR) and pitch angle.

For MHK turbines equipped with pitch actuators, those turbine can can use control schemes similar to those used by wind turbines, which reduce the Cp surface by increasing the pitch angle (black). For MHK turbines without pitch actuation, we provide a few control methods for controlling the power of the turbine using only torque control. Overspeed control increases the TSR along the fixed pitch line on the Cp surface below (blue circle) by decreasing the generator torque. Underspeed control decreases the TSR (red circle) by increasing the torque.



8.2 Over/Underspeed Reference Setpoints

The steady state generator-speed setpoints are determined by the Cp contour. Overspeed achieves up to 3x rated speed, which has additional consequences for blade loads (e.g., cavitation)



Torque setpoints ($\bar{\tau}$) determined by constant-power relationship $\bar{\tau} = \frac{P_{rated}}{\bar{\omega}}$, where P_{rated} is the rated power and $\bar{\omega}$ is the steady state generator speed.

In Region 3, the relationship between torque and speed are nonmonotic. Thus, more careful reference control design is required for managing the transition region. There are examples in the literature for saturation/smoothing during the transition region.

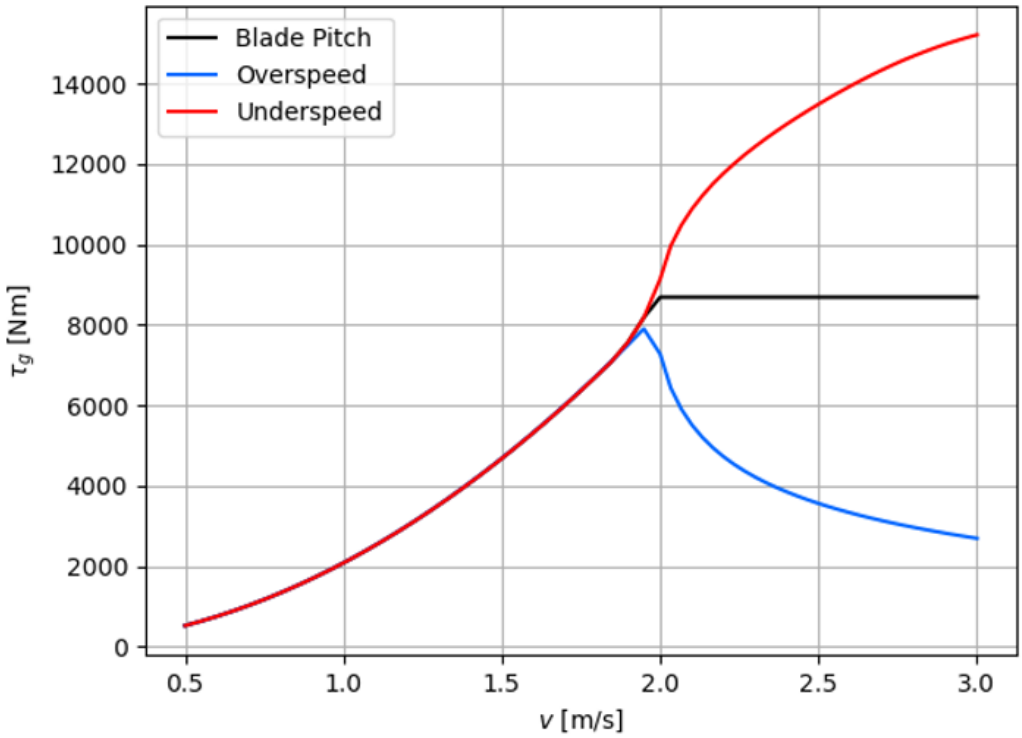
8.3 Over/Underspeed Dynamics

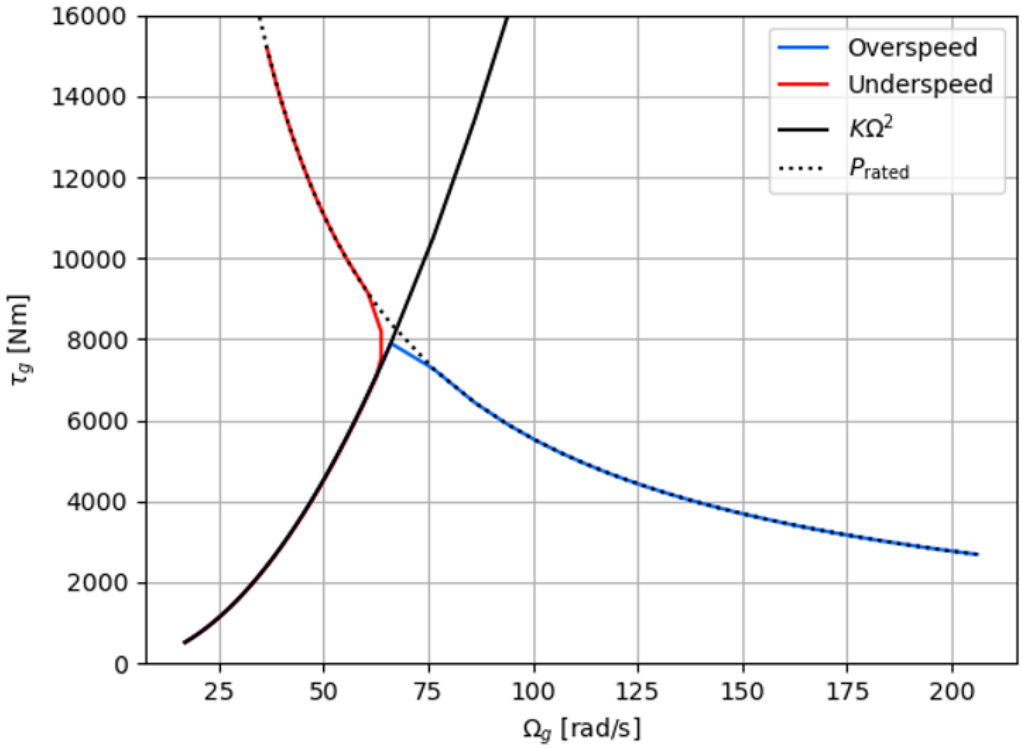
At each operating point, the sensitivity is computed using the gradients of the Cp surface. The first-order system decay rate is represented by a single pole on the real axis: more negative means more rapidly stable (positive means unstable). Underspeed set points are open-loop unstable at high flow speeds.

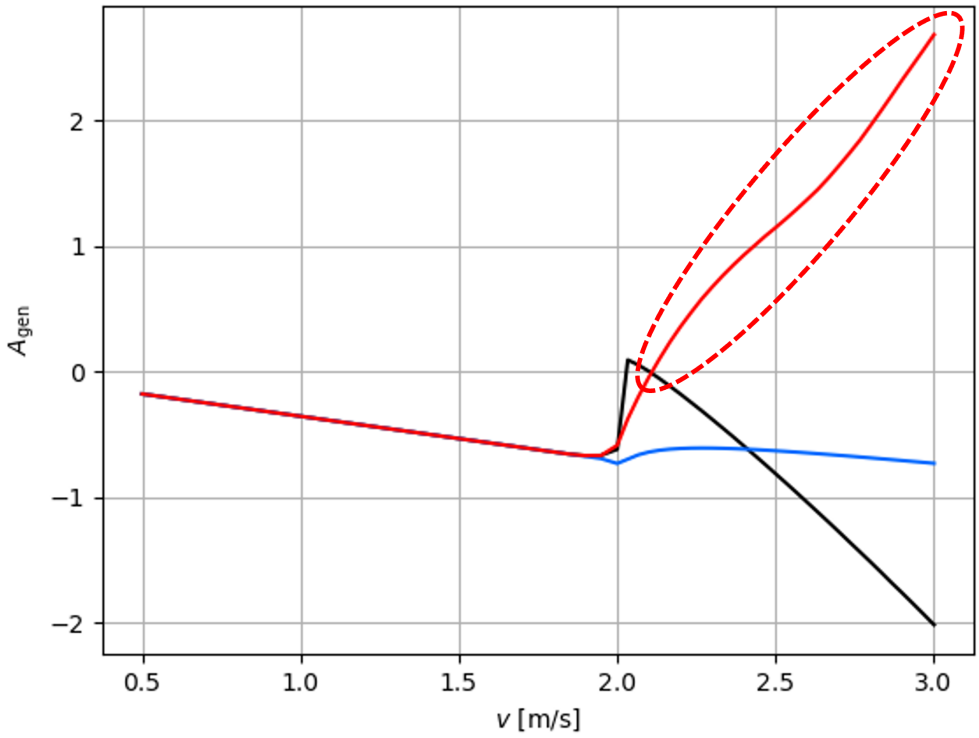
@DS: what is shown here?

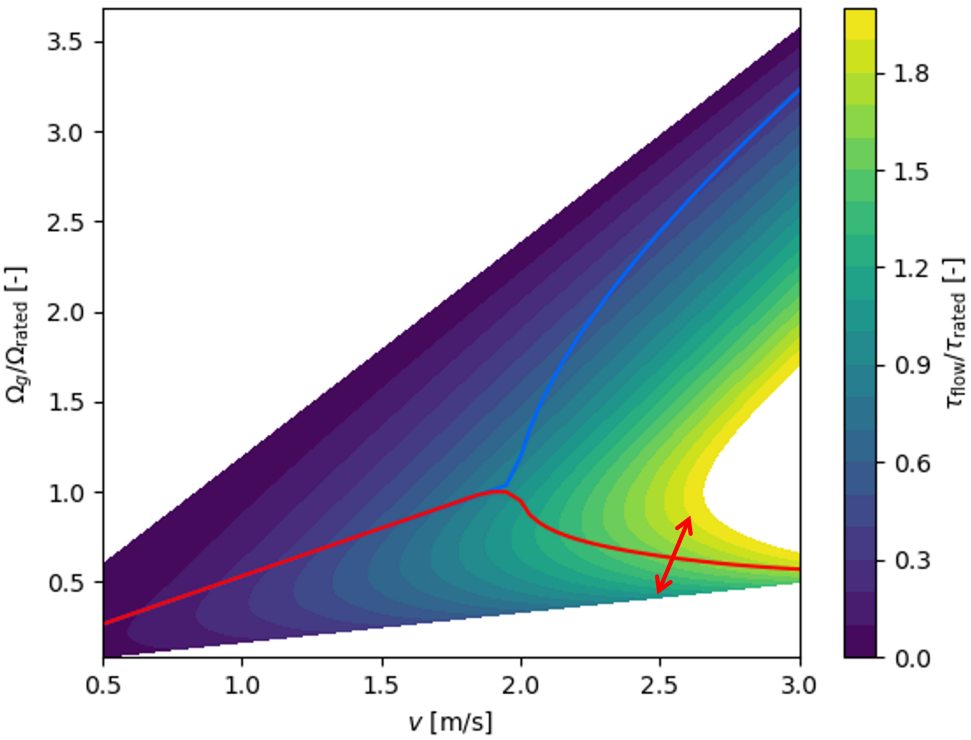
8.4 Fixed-Blade-Pitch (FBP) Control

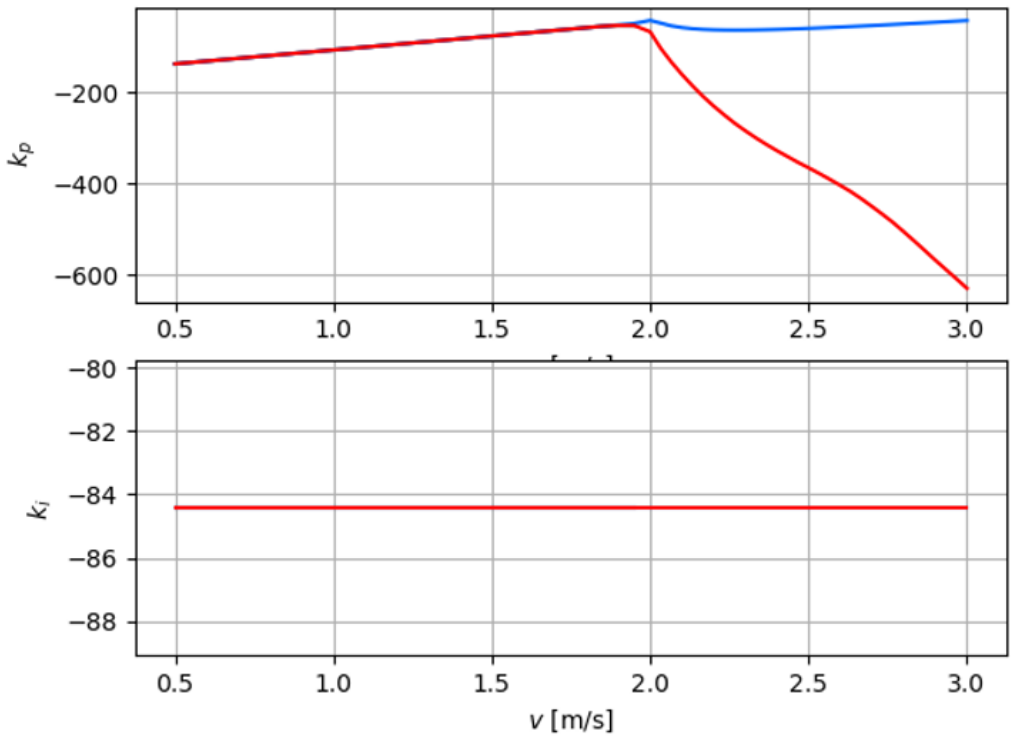
At each setpoint, the torque controller gains are determined using the process as the normal torque control in ROSCO. High magnitude gains are required to compensate for the open-loop instability of the underspeed system (red).


Blade Pitch
Overspeed
Underspeed
14000
12000
10000
8000
6000
4000
2000
0
τ_g [Nm]
0.5
1.0
1.5
2.0
2.5
3.0
v [m/s]


Overspeed
Underspeed
$K\Omega^2$
P_{rated}
16000
14000
12000
10000
8000
6000
4000
2000
0
τ_g [Nm]
25
50
75
100
125
150
175
200
Ω_g [rad/s]


2
1
0
−1
−2
A_{gen}
0.5
1.0
1.5
2.0
2.5
3.0
v [m/s]


3.5
3.0
2.5
2.0
1.5
1.0
0.5
Ω_g/Ω_{rated} [-]
0.5
1.0
1.5
2.0
2.5
3.0
v [m/s]
1.8
1.5
1.2
0.9
0.6
0.3
0.0
τ_{flow}/τ_{rated} [-]



8.5 Alternate Region 3 Operating Schedules

Using the ROSCO toolbox, we enable the user to determine their own operational power curve, besides a constant rated power.

The alternative power curves result in different speed and torque set points (dashed lines represent underspeed, solid overspeed).

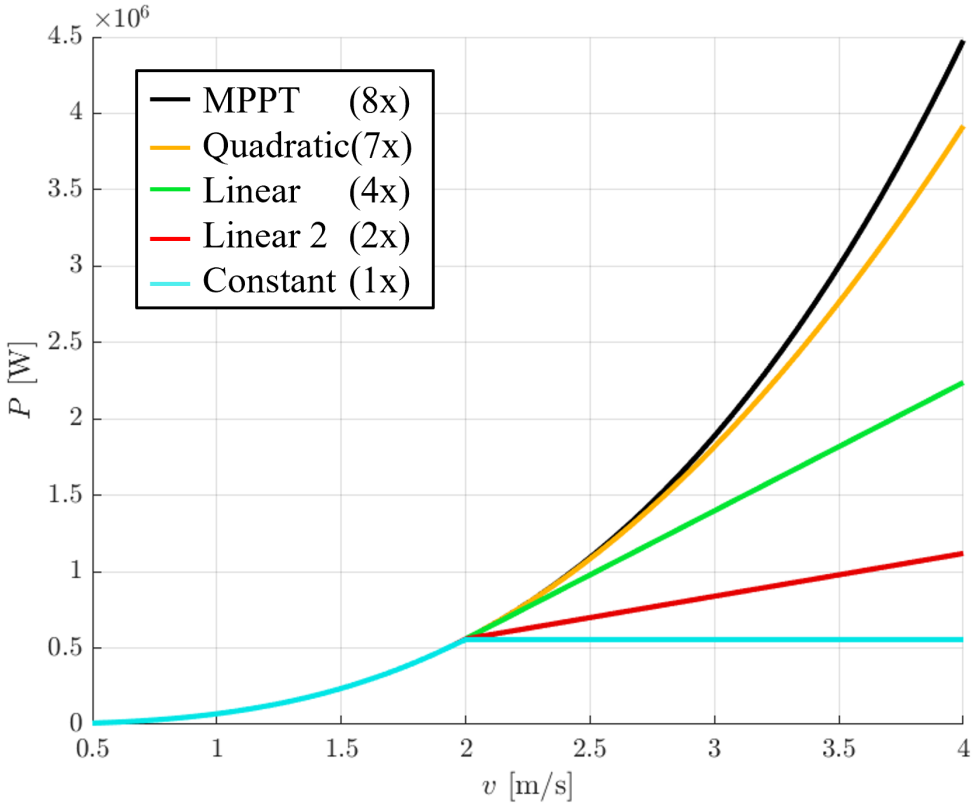
The stability can be represented by the sensitivity $\frac{d\tau}{d\Omega}$. Values less than 0 are open-loop stable.

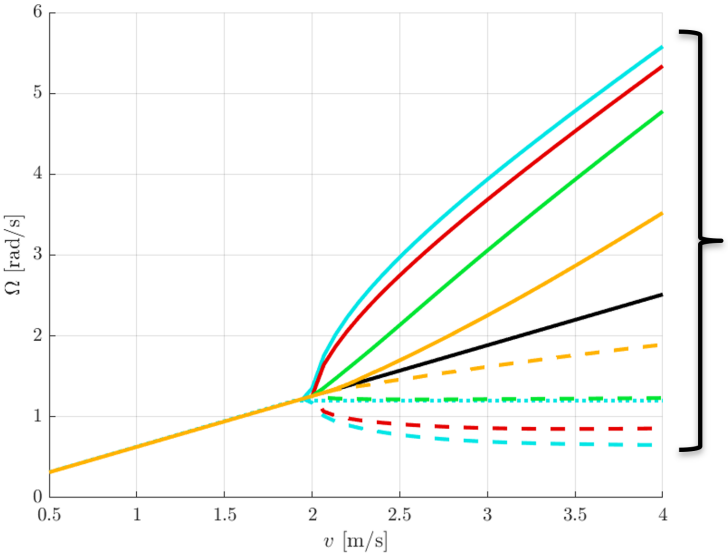
The power curve selection also impacts the rotor thrust (F). Underspeed control and lower power generally results in lower thrust.

8.6 Toolbox Implementation

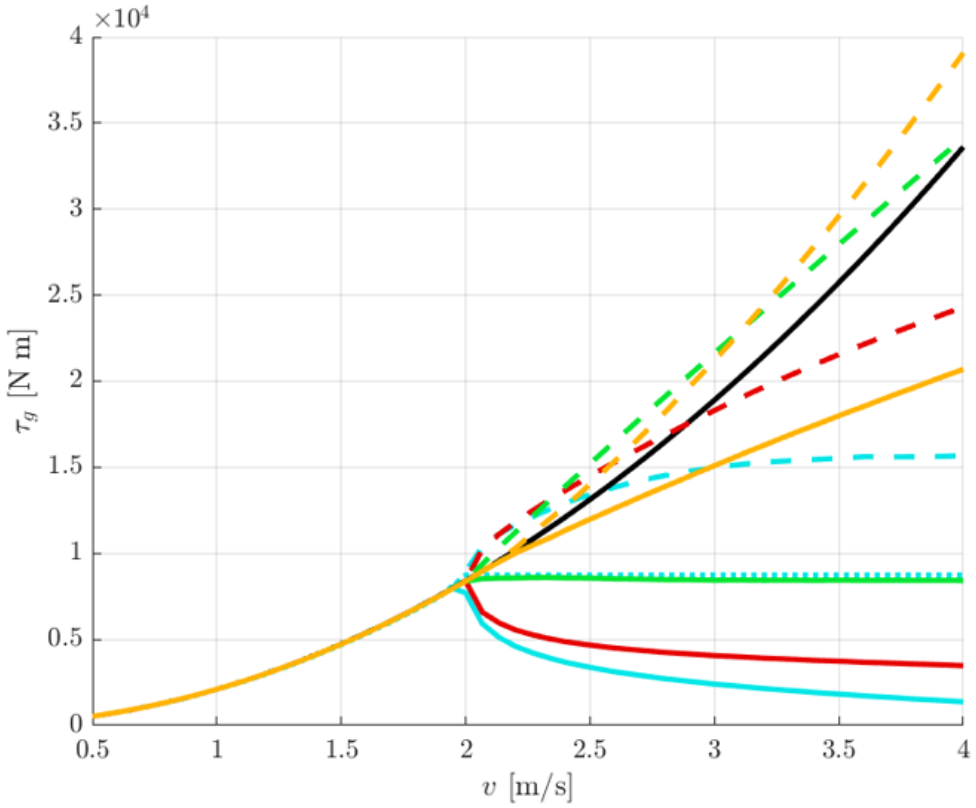
The ROSCO toolbox works by determining the speed and torque set points required to operate at a TSR and Cp for the desired power.

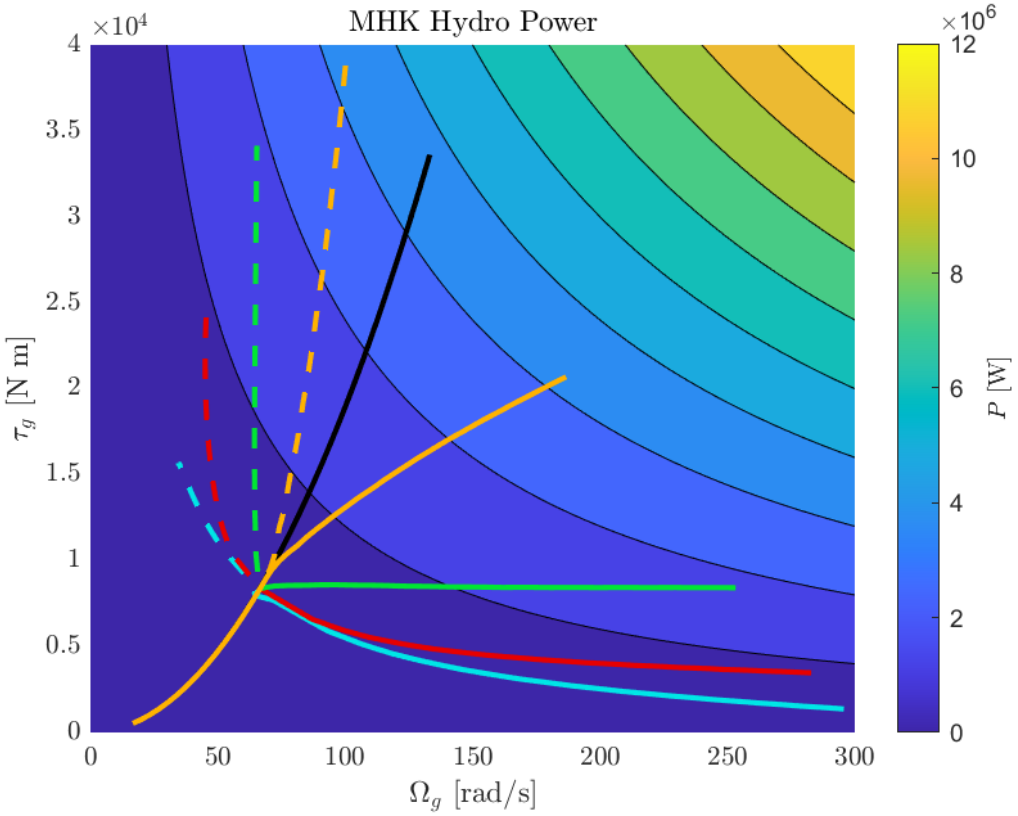
The following inputs to the ROSCO tuning yaml will generate DISCON inputs to ROSCO.

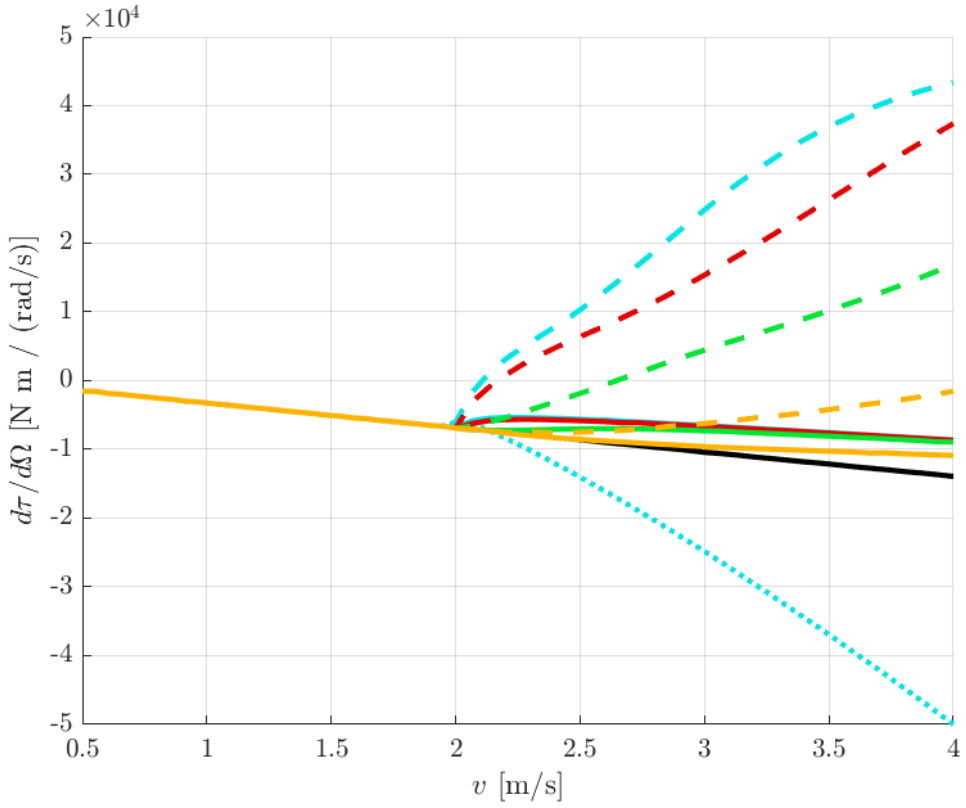


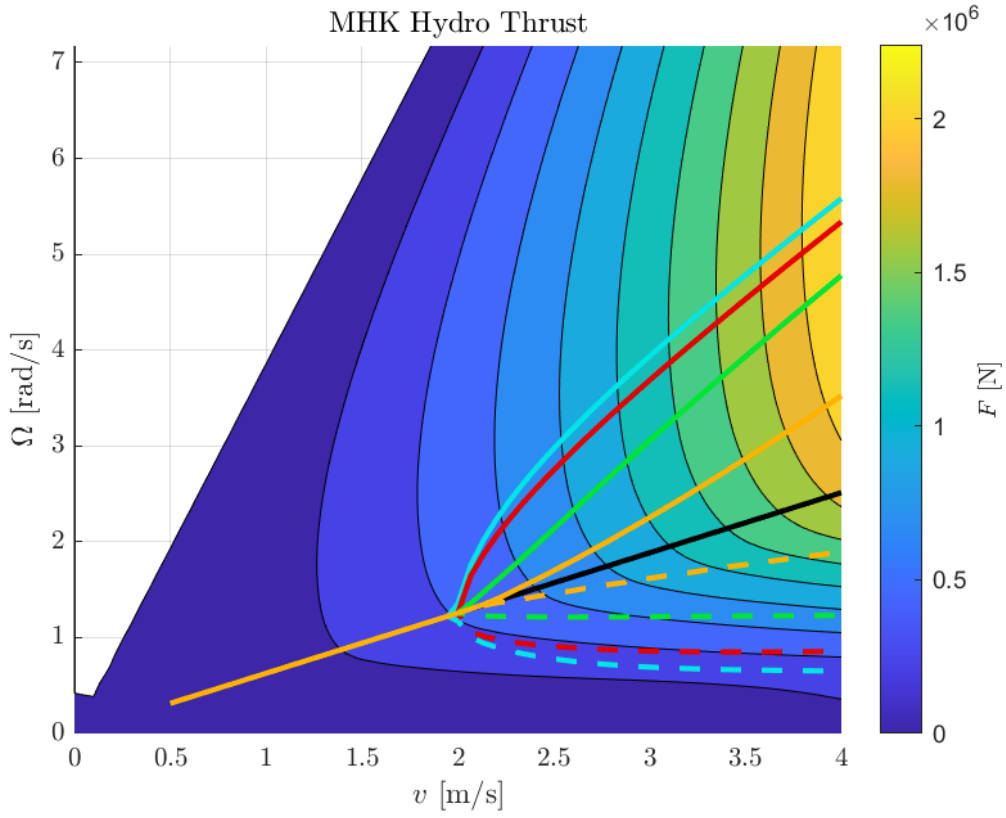


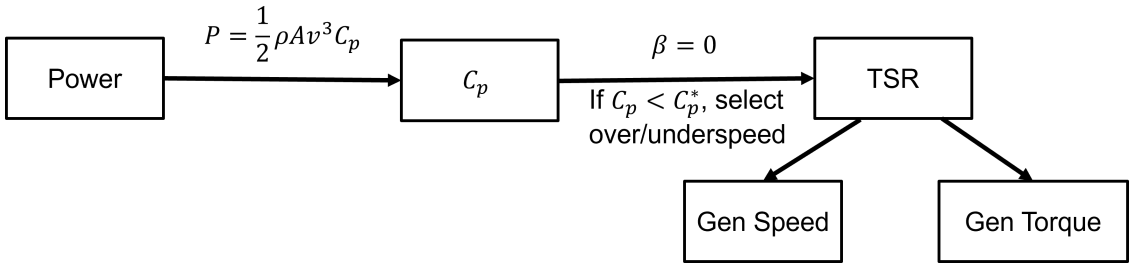
Speed setpoints are within the envelope between over/underspeed const power


×10^4
4
3.5
3
2.5
2
1.5
1
0.5
0
τ_g [N m]
0.5
1
1.5
2
2.5
3
3.5
4
v [m/s]


MHK Hydro Power
×10^4
4
3.5
3
2.5
2
1.5
1
0.5
0
τ_g [N m]
0
50
100
150
200
250
300
Ω_g [rad/s]
×10^6
12
10
8
6
4
2
0
P [W]


×10^4
5
4
3
2
1
0
-1
-2
-3
-4
-5
0.5
1
1.5
2
2.5
3
3.5
4
dτ/dΩ [N m / (rad/s)]
v [m/s]


MHK Hydro Thrust
×10^6
2
1.5
1
0.5
0
F [N]
7
6
5
4
3
2
1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
Ω [rad/s]
v [m/s]



Parameter	Description
VS_FBP	FBP Control Mode (0 = variable pitch, 1 = constant power overspeed (nonlinear), 2 = WSE-lookup reference tracking, 3 = torque-lookup reference tracking)
FBP_speed_r	Over/underspeed mode (0 = underspeed, 1 = overspeed)
FBP_power_	Normalized or exact power curve values (0 = relative to rated, 1 = exact)
FBP_U	Flow speed setpoints for power curve lookup table
FBP_P	Power curve lookup table

Note that the ROSCO input schema (*ROSCO_Toolbox tuning .yaml*) contains the latest input definitions.

8.7 ROSCO Implementation

The following DISCON parameters are generated using the ROSCO toolbox, or can be determined directly in the DISCON.IN file.

Parameter	Description
VS_FBP	FBP Control Mode (0 = variable pitch, 1 = constant power overspeed (nonlinear), 2 = WSE-lookup reference tracking, 3 = torque-lookup reference tracking)
VS_FBP_n	Number of values in operating schedule lookup table
VS_FBP_U	Flow speed operating points in lookup table
VS_FBP_Or	Generator speed operating points in lookup table
VS_FBP_Ta	Generator torque operating points in lookup table

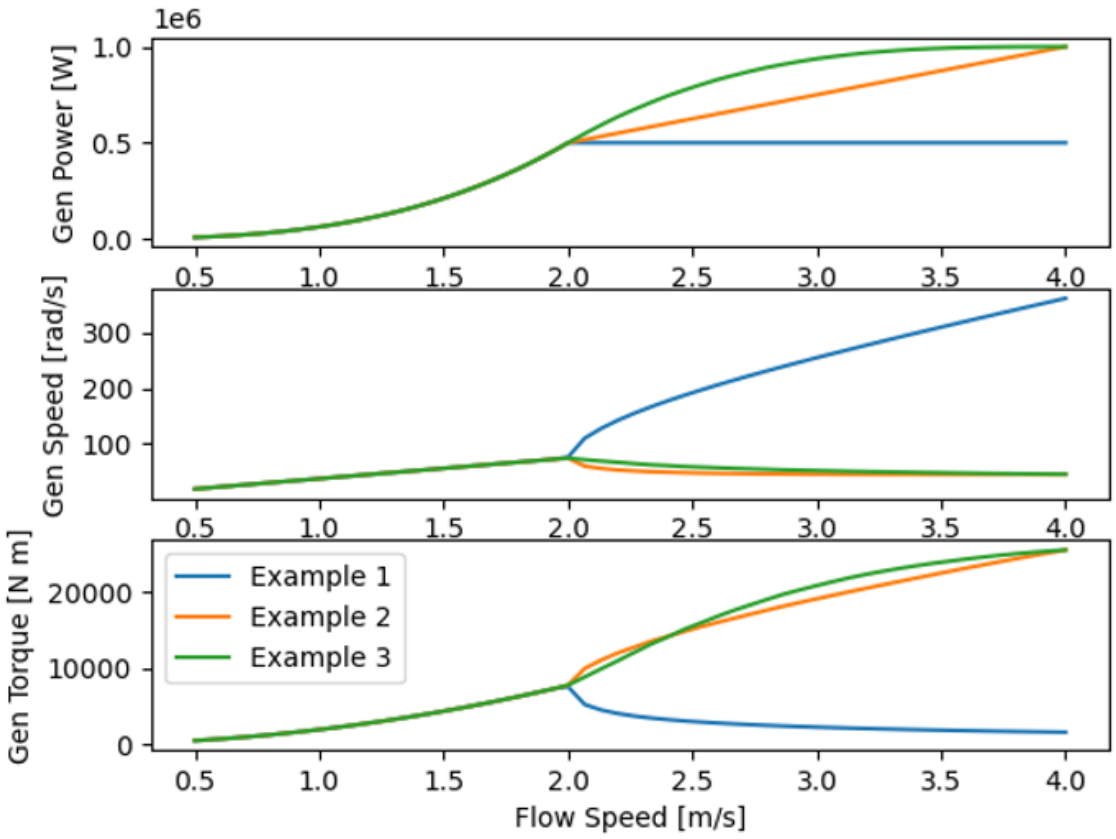
Note that the ROSCO input schema (*ROSCO_Toolbox tuning .yaml*) contains the latest input definitions (under `controller_params`, `DISCON`).

8.8 Simulation Verification

A handful of example controller case studies have been developed using the RM1 marine turbine to showcase the implemented features of FBP control. These configurations are

- Example 1: Overspeed, constant power
- Example 2: Underspeed, torque-based reference tracking
- Example 3: Underspeed, WSE-based reference tracking

For each example, a power curve is defined and input to the ROSCO toolbox to generate operating schedules and auto-tune the gains used by the torque controller. The operating schedules for generator power, speed, and torque for each example test case are shown in the following figure.



Each example controller is then simulated with the RM1 marine turbine model using OpenFAST in both steady and turbulent inflow. The steady-state performance of each controller is compared to the operating schedules generated by the ROSCO toolbox. The turbulent inflow uses the intensity shown in the following figure.

8.8.1 Example 1

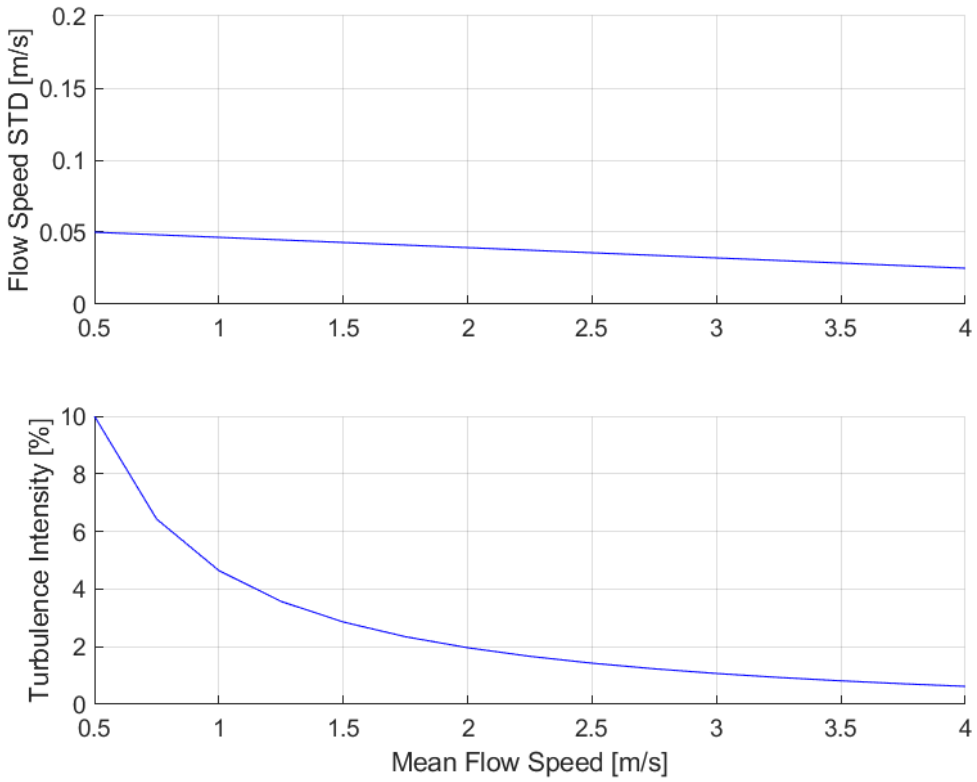
The first example test case uses the naturally stable nonlinear feedback control law. This controller is confined to operating in the constant power, overspeed configuration. The explicit (non-reference-tracking) control law is analogous to the *kOmega^2* control law sometimes used in Region 2 for wind and marine turbines.

This controller has the best power tracking in Region 3, but it only allows constant power. The power-focused feedback approach accommodates offsets in equilibrium speed and torque made by inaccuracies in the simplified tuning model.

8.8.2 Example 2

The second example test case

- Region-2 mode: TSR-tracking with torque-based reference
- Region-3 FBP mode: reference tracking with torque lookup
- The power curve may be arbitrarily specified, but should be a nondecreasing function so that the torque schedule is monotonically increasing
- Linearly increasing power in Region 3, up to 2x rated
- Power curve must be set so that torque schedule is monotonic
- Decent power tracking

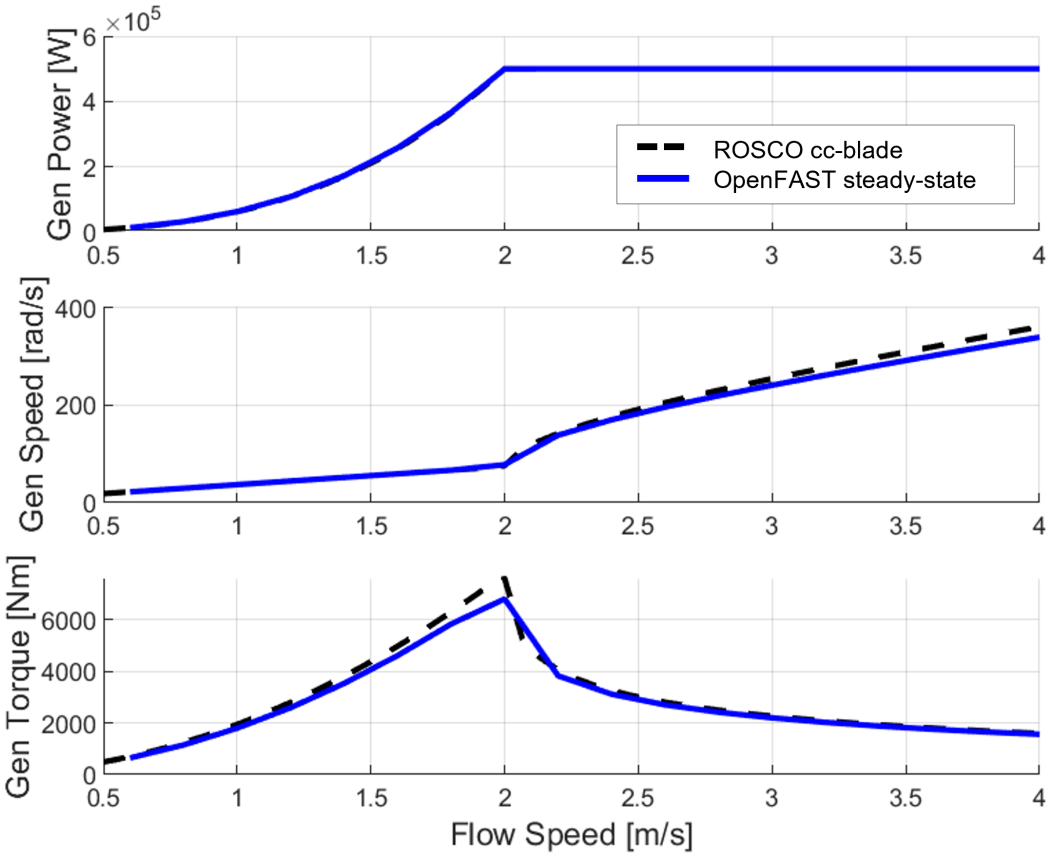


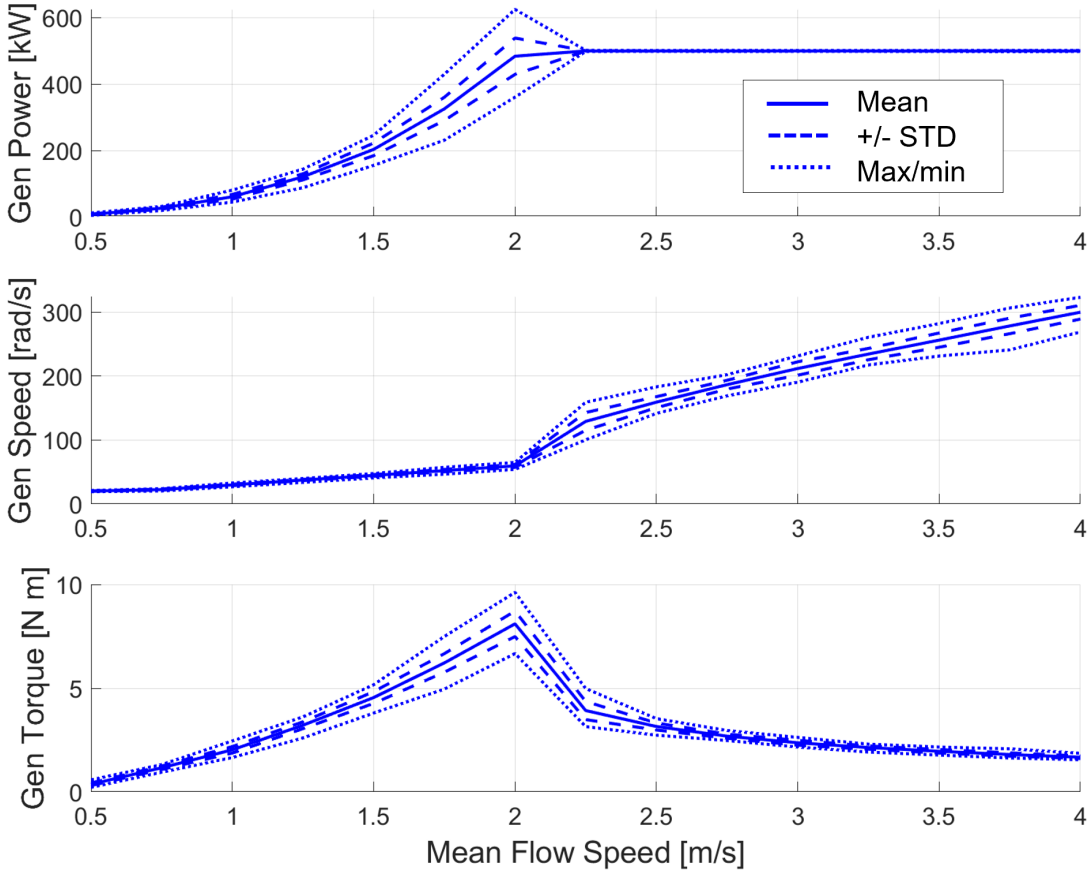
- May have some misalignment with flow speed setpoint
- Accommodates some offsets in torque and speed
- Should be combined with reference-tracking controller in Region 2

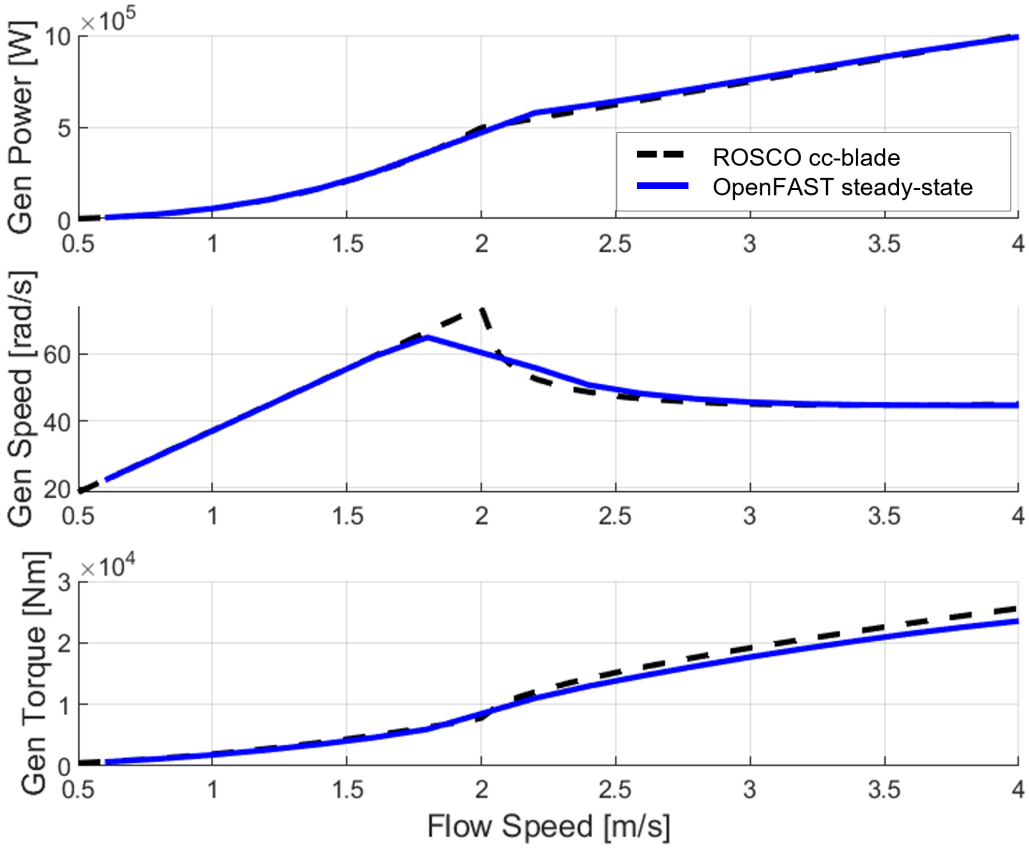
8.8.3 Example 3

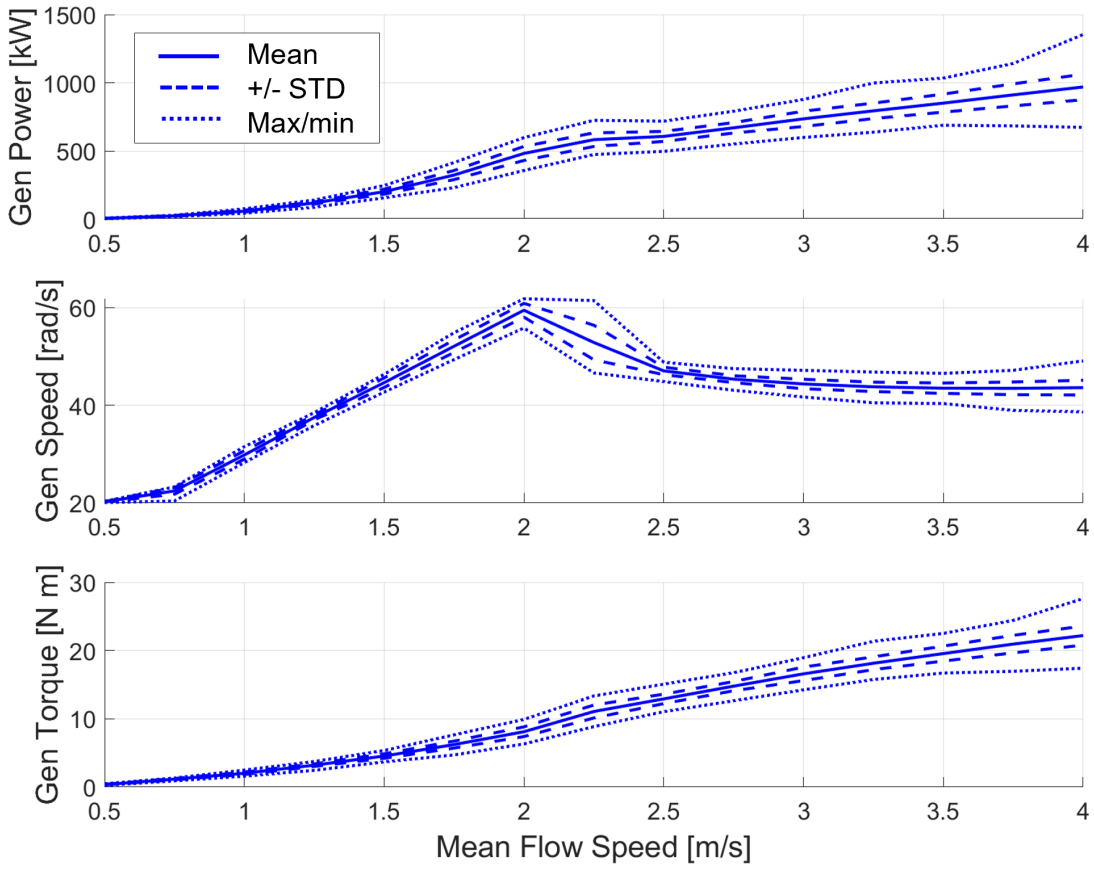
The third example test case

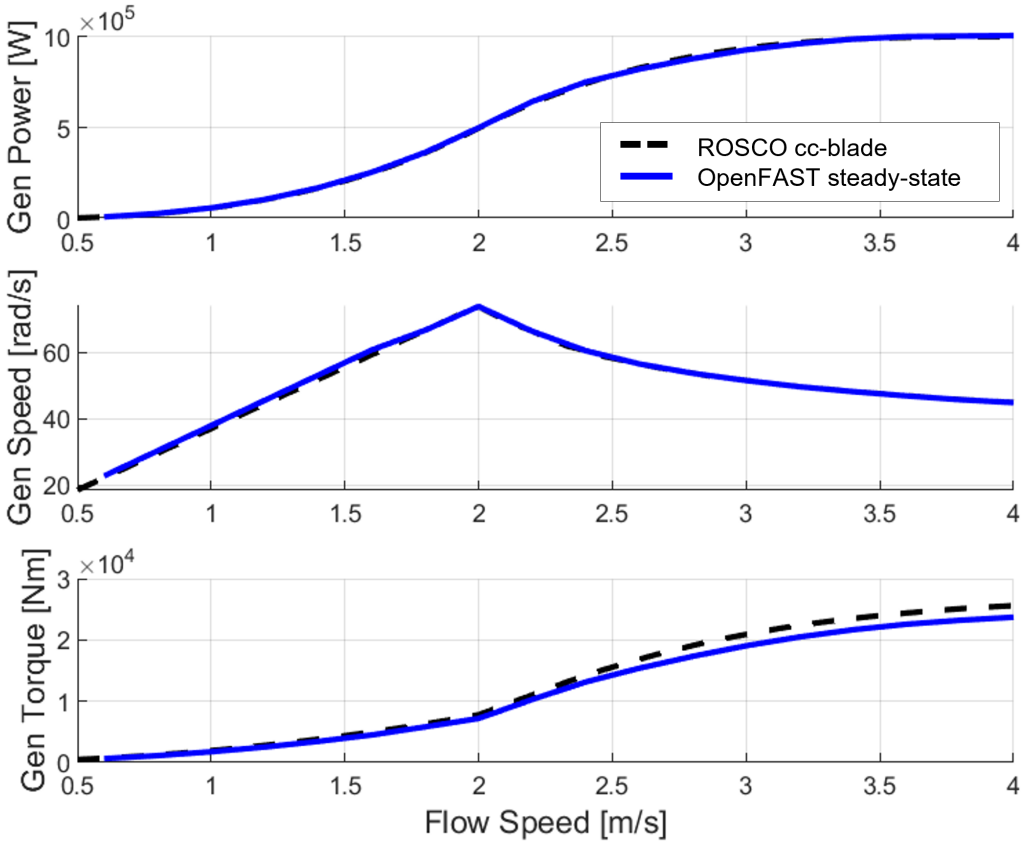
- Region-2 mode: TSR-tracking with WSE-based reference
- Region-3 FBP mode: reference tracking with WSE lookup
- The power curve can be completely arbitrarily specified
- Smoothly increasing power curve in Region 3
- Would allow arbitrarily increasing or decreasing
- Best gen speed tracking
- May offset equilibrium torque leading to power curve error
- Should be combined with reference-tracking controller in Region 2

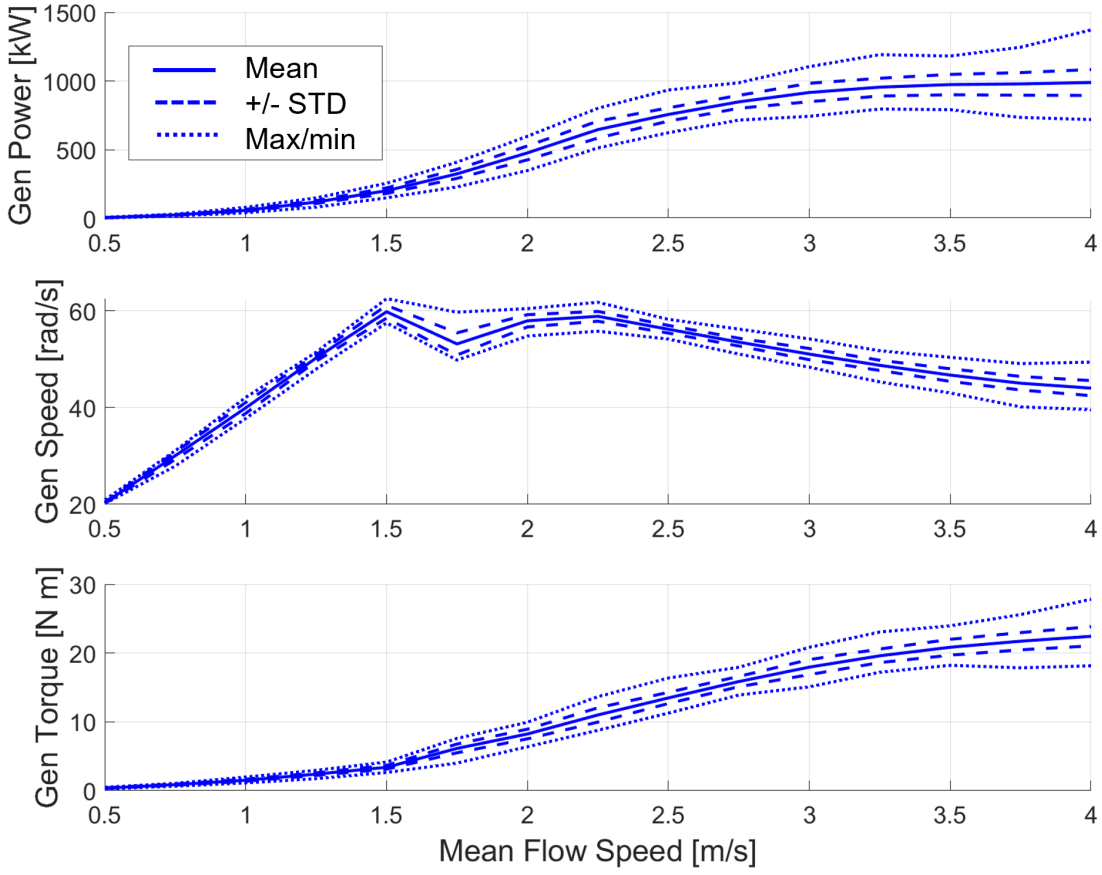

×10^5
Gen Power [W]
6
4
2
0
0.5
1
1.5
2
2.5
3
3.5
4
ROSCO cc-blade
OpenFAST steady-state
Gen Speed [rad/s]
400
200
0
Gen Torque [Nm]
6000
4000
2000
0
Flow Speed [m/s]


Gen Power [kW]
600
400
200
0
Mean
+/- STD
Max/min
Gen Speed [rad/s]
300
200
100
0
Gen Torque [N m]
10
5
0
0.5
1
1.5
2
2.5
3
3.5
4
Mean Flow Speed [m/s]


Gen Power [W]
×10^5
10
5
0
ROSCO cc-blade
OpenFAST steady-state
Gen Speed [rad/s]
60
40
20
Gen Torque [Nm]
×10^4
3
2
1
0
0.5
1
1.5
2
2.5
3
3.5
4
Flow Speed [m/s]


Gen Power [kW]
1500
1000
500
0
Mean
+/- STD
Max/min
0.5
1
1.5
2
2.5
3
3.5
4
Gen Speed [rad/s]
60
40
20
Gen Torque [N m]
30
20
10
0
Mean Flow Speed [m/s]


Gen Power [W]
×10^5
10
5
0
ROSCO cc-blade
OpenFAST steady-state
Gen Speed [rad/s]
60
40
20
Gen Torque [Nm]
×10^4
3
2
1
0
0.5
1
1.5
2
2.5
3
3.5
4
Flow Speed [m/s]


Gen Power [kW]
1500
1000
500
0
Mean
+/- STD
Max/min
0.5
1
1.5
2
2.5
3
3.5
4
Gen Speed [rad/s]
60
40
20
Gen Torque [N m]
30
20
10
0
Mean Flow Speed [m/s]

8.9 Recommendations

FBP control is well suited to marine turbines without blade pitch actuators. In certain applications, the ability to follow a generic power curve with a limit actuation capability is more advantageous than using variable-blade-pitch (VBP) control. VBP control allows constant-power operation in Region 3 matched with constant speed and torque for a flat operating schedule. Pitch-actuated turbines also experience smaller blade loads in Region 3. The FBP approach satisfies applications in which the cost and complexity of the actuators themselves are prohibitive.

Generic user input allows flexibility for variety of applications.

FBP controller implementation in ROSCO with auto-tuning and automatic generation of operating schedule to follow power curve.

Because the Region-2 and Region-3 controllers utilize the same actuator, the transition region is markedly different than what is required for a VBP Region-3 controller.

CHAPTER

NINE

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.

9.1 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.

9.2 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/
```

9.3 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.

9.4 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.

9.5 Adding Inputs to ROSCO

9.5.1 ROSCO controller

- **Add control parameters to the registry: See the rosco_types.yaml file.**
  - Any `LocalVariables` that you want to log are also in the registry: see here.
  - The convention is that `LocalVariables` update every timestep, whereas `ControlParameters` do not change.
- **Regenerate Fortran variable declarations and logging automatically:** Run the registry script to regenerate `ROSCO_Types.f90` and `ROSCO_IO.f90`: write_registry.py.
- **Parse inputs in Fortran:** Update `ReadSetParameters`: see ReadSetParameters.f90.

9.5.2 ROSCO toolbox

- **Add to the DISCON schema:** Update toolbox_schema.yaml.
- **Add to the DISCON file writer:** Update utilities.py.

9.5.3 Update DISCON inputs

- See the script: update_rosco_discons.py

9.6 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

TEN

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:

10.1 Bladed versions 4.6 to current (4.12)

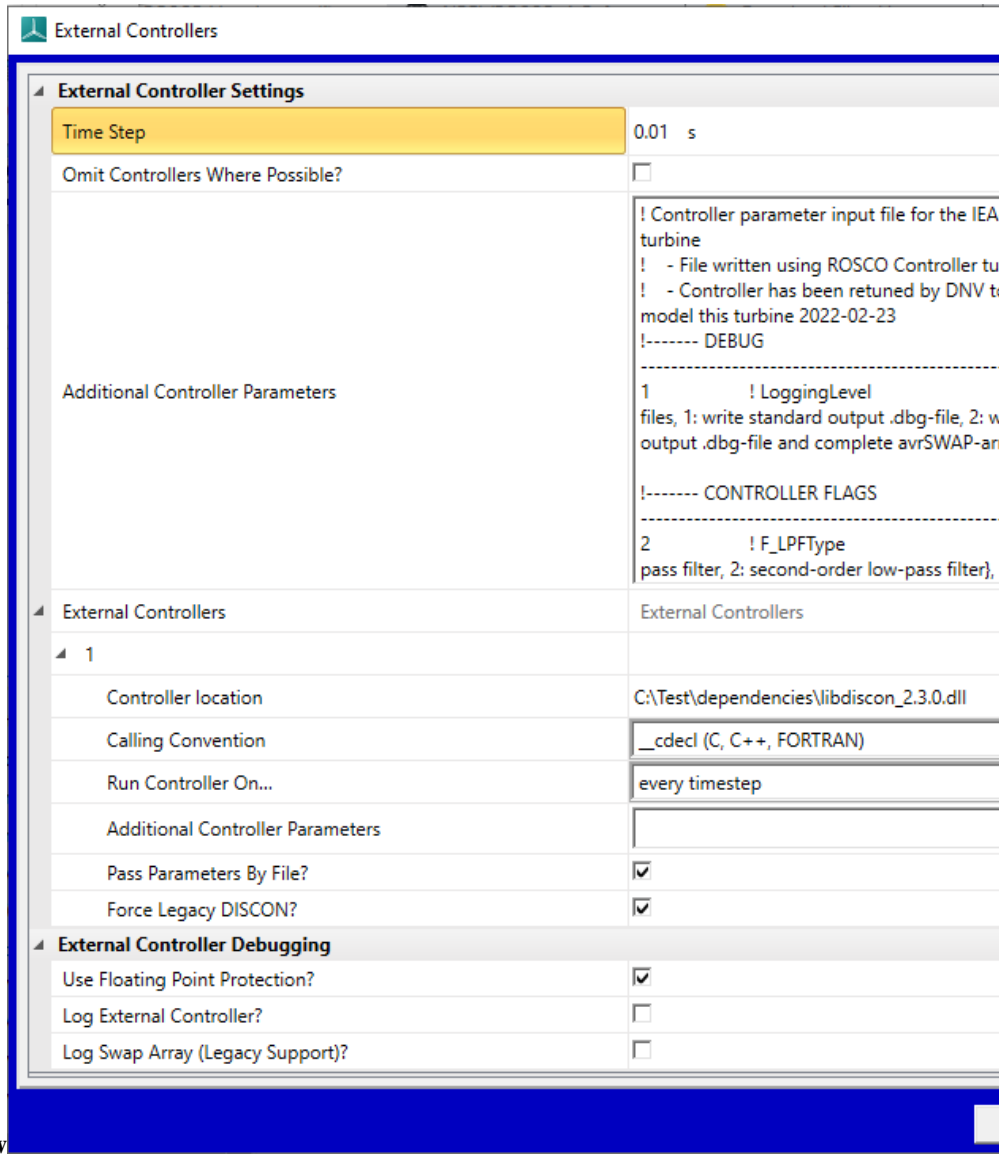
In the Bladed External Controller dialog, fill in the fields as follows:

- *'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



10.2 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

ELEVEN

LICENSE

PYTHON MODULE INDEX

INDEX

Symbols

M