Delft Research Controller

An open-source and community-driven wind turbine baseline controller

Mulders, S. P.; Van Wingerden, J. W.

DOI
10.1088/1742-6596/1037/3/032009

Publication date
2018

Document Version
Final published version

Published in
Journal of Physics: Conference Series

Journal of Physics: Conference Series

PAPER • **OPEN ACCESS**

Delft Research Controller: an open-source and community-driven wind turbine baseline controller

To cite this article: SP Mulders and JW van Wingerden 2018 *J. Phys.: Conf. Ser.* **1037** 032009

IOP Conf. Series: Journal of Physics: Conf. Series **1037** (2018) 032009 doi:10.1088/1742-6596/1037/3/032009

Delft Research Controller: an open-source and community-driven wind turbine baseline controller

SP Mulders and JW van Wingerden

Delft Center for Systems and Control, Faculty of Mechanical Engineering, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands

E-mail: S.P.Mulders@tudelft.nl, J.W.vanWingerden@tudelft.nl

Abstract. Wind energy research groups from various disciplines generally use self-developed baseline wind turbine control implementations and tunings, which complicates the evaluation and comparison of new control algorithms. To solve this problem, the Delft Research Controller (DRC) provides an open, modular and fully adaptable baseline wind turbine controller to the scientific community. New control implementations can be added to the existing baseline controller, and in this way, convenient assessments of the proposed algorithms is possible. Because of the open character and modular set-up, scientists are able to collaborate and contribute in making continuous improvements to the code. The DRC is being developed in Fortran and uses the Bladed-style DISCON controller interface. The compiled controller is configured by a single control settings parameter file, and can work with any wind turbine model and simulation software using the DISCON interface. Baseline parameter files are supplied for the NREL 5-MW and DTU 10-MW reference wind turbines.

1. Introduction

The existence of reference models and baseline cases is a crucial aspect in the scientific community, as it allows for convenient and fair evaluation of proposed innovations. In the wind turbine community, the National Renewable Energy Laboratory (NREL) offshore 5-MW baseline wind turbine is a fictive but fully defined reference wind turbine (RWT) model [1], and is actively used in the scientific field. To accommodate the next step in enlarging the size and rated power of offshore wind turbines, the Technical University of Denmark (DTU) provides a 10-MW reference wind turbine model [2]. The DTU10MW model is developed in cooperation with Vestas Wind Systems. Its design is mainly focused on setting a reference for next generation rotors, with good aerodynamic performance at a relative low weight.

Besides reference models, wind turbine simulation software is also largely standardized. The industry standard, commercial and certified high-fidelity wind turbine simulation package is Bladed by DNV GL [3]. On the other hand, an open-source aeroelastic package for simulating horizontal-axis wind turbines is FAST, which was up until recently actively developed and maintained by NREL. However, a shift towards community-driven software development is seen in the scientific field, and OpenFAST is established with the FAST v8 code as its starting point [4]. The goal of OpenFAST is being a community model, with users and developers from research laboratories, academia and industry improving software quality and accelerating development.

In contrast to the previously mentioned reference models and simulation software, no clear choice of a baseline wind turbine controller currently exists that is easy to use, modular and

extendable. Wind energy research groups from various disciplines generally use self-developed baseline control implementations and tunings, of which the source code is rarely available. This negatively impacts the ability to compare results from different research projects or groups. It has to be noted that NREL provides an open-source controller for its NREL5MW reference wind turbine [1]. However, this controller is limited in functionality and inconvenient to extend or interchange between distinct wind turbine models as functionality, turbine parameters and controller tunings are hard-coded in a single source file. DTU also provides an internally developed controller for their DTU 10-MW RWT [5] of which the source code is available, but the development is not driven by a broad community.

To this end, the Data Driven Control wind energy research group from Delft University of Technology started the initiative to develop an open-source and community-driven wind turbine baseline controller. A design specification was that the controller should be generally applicable to all turbine models defined in simulation software that uses the Bladed-style DISCON controller interface [6], such as OpenFAST, Bladed or HAWC2. Also, a convenient way of configuring the controller should be present, without editing the source code and thus the need for recompilation. With these goals in mind, the foundations of a baseline wind turbine controller have recently been laid out, and is dubbed the Delft Research Controller (DRC). For consistency with OpenFAST, the DRC is being developed in the Fortran programming language (free-form) [7]. The modular and open character allows scientists to collaborate and contribute in making continuous improvements to the code. The DRC is provided with a toolbox consisting of regularly used (control) functions and filters to allow rapid development and implementation of new contributions. Remarkable functionality of the DRC is the ability to activate and switch between conventional yaw-rate [8] and novel yaw-by-IPC control implementations [9].

The main contribution of this paper is to provide a comprehensive description of the DRC. The organization of this paper is as follows. In Section 2, an overview of the DRC is given including the built-in function and filter modules. These components are used to make up the baseline torque, (individual) pitch and yaw controllers, described in Section 3. In Section 4, simulations of the DTU10MW controlled by the DRC showcase the performance of various control options. In Section 4, conclusions are drawn and an outlook is given on future additions.

2. DRC components

This section gives an overview and description of the various components included in the DRC. A general overview of the controller is given in Section 2.1, after which the included filters and functions are described in Sections 2.2 and 2.3, respectively. In the remainder of this paper, internal controller variables, data-types, modules and subroutines are presented in a `typewriter font`, whereas parameters defined in the controller configuration (parameter)file are <u>`underlined`</u>.

2.1. Overview of the DRC

This section gives an overview of the DRC, of which the overall working principle is presented in Figure 1. The DRC is set up such that only a single parameter file `ControllerParameters.in` for each wind turbine model is required to define the complete control system. This feature removes the need for repetitive recompilation of the controller under a change in control settings. To prevent computational errors related to unit conversions, all variables and constants are defined in SI units. The input file allows defining different debugging levels: setting <u>`LoggingLevel`</u> to `0` omits writing a log file, defining it as `1` writes specified variables to a `.dbg` file, and configuring the parameter to `2` additionally writes the complete `avrSWAP`-array to a `.dbg2` file.

The DRC consists of multiple modules containing commonly used functions and subroutines. The `Constants.f90` module contains regularly used non-varying parameters, such as unit

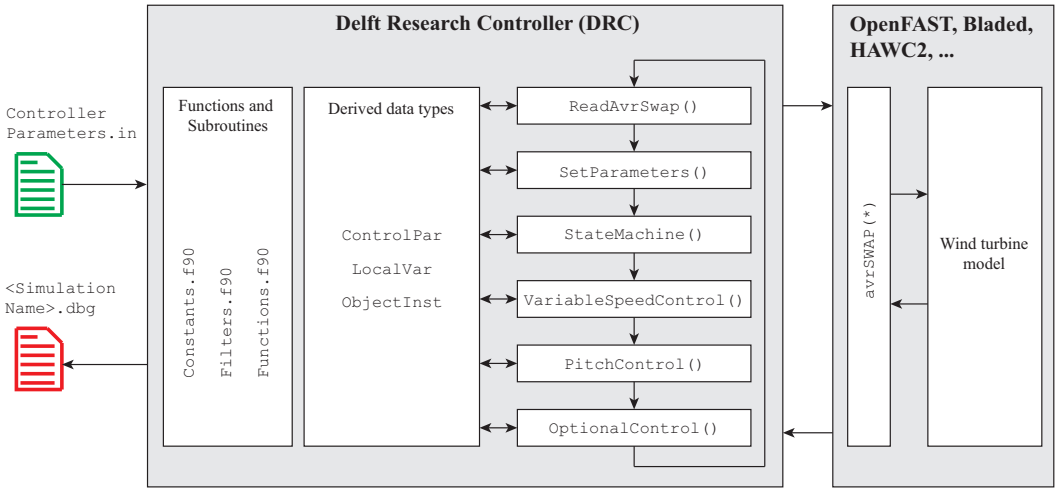

Figure 1. A schematic of the DRC architecture for wind turbine control. The controller exchanges data using the so-called the DISCON external controller interface via the avrSWAP-array. The DRC is completely parameterized by a single configuration file, and writes debug information to a log file when desired.

conversions and physical constants. The `Functions.f90` module contains general procedures, and the `Filters.f90` module consists out of discretized signal filters.

The DRC uses derived data types to store parameters and variables in a centralized manner. A derived data type has similarities to C and MATLAB structures, or C++ classes. The values defined in the `ControllerParamameters.in` file are loaded in the `ControlPar` data type and are stored as read-only. All local controller variables are stored in the `LocalVar` data type, such that they are accessible among the different control subroutines.

The DRC executes function calls in a fixed sequence during each control iteration. The DRC reads the control in- and output `avrSWAP`-array by calling its `ReadAvrSwap()` subroutine, and the `SetParameters()` subroutine evaluates required controller parameters and performs value assertions. Next, before calling any controller, the state-machine subroutine `StateMachine()` determines the state of the turbine, and this information is used by the controllers to perform corresponding control actions. To enable the Variable-Speed Variable-Pitch (VSVP) control strategy, torque and pitch control subroutines (`VariableSpeedControl()` and `PitchControl()`) are implemented. Optional controllers are executed after the two before-mentioned controllers.

It would be particularly interesting to enable distributed control of wind turbines in a wind farm set-up [10]. The Simulator for Offshore Wind Farm Applications (SOWFA) already has a build in super-controller that is able to communicate with local wind turbine controllers [11]. In later stages, this coupling between the DRC and SOWFA will be made possible.

The DRC source code is publicly available under terms of the GNU Public License version 3 as a repository on the Data-Driven Control GitHub website

`https://github.com/TUDelft-DataDrivenControl/DRC_Fortran`

An advantage of using a public repository is the ability to share the controller by only referring to the Git commit hash (controller version) and parameter file. This improves reproducibility of the performed research, as all changes between commits remain accessible.

Along with the source code, a makefile and a Visual Studio project are supplied for compilation with `gfortran` [12] or `Intel Visual Fortran` [13] under Linux and/or Windows,

for creation of `.so` and `.dll` library files. Baseline parameter configuration files are supplied for the NREL5MW and DTU 10-MW reference wind turbines.

2.2. Filter module

The DRC comes with a collection of frequently used filters, which are contained in the filter module `Filters.f90`. In this section, the continuous time representations of the included filters are discussed. The filters included are discretized using the bilinear transformation, also known as Tustin's method, which is defined by

$$s = \frac{2}{\mathtt{DT}} \frac{1 - z^{-1}}{1 + z^{-1}}, \tag{1}$$

where `DT` is the sampling time and z^{-1} the discrete-time unit delay operator. This transformation maps every point of the frequency response of the continuous-time filter to a corresponding point in the frequency response of the discrete-time filter [14]. The filters are not bound to a predefined sampling time, as this variable is taken as an input from the simulation. The general function-call syntax is given by `filter(InputSignal, DT, Opt1, Opt2, ..., iStatus, reset, inst)`. To avoid transients, the inputs `iStatus` and a `reset` are used to reinitialize the filter states (initial condition) to the current input signal. All functions return a single real filtered output signal, with a default steady-state gain of 0 dB.

Fortran is not an object-oriented language, which would be especially convenient when working with multiple instances of the same subroutine. For example, one might want to use a certain filter for different purposes without redefinition. To solve this problem, the DRC incorporates an implementation to create multiple instances of functions and subroutines and stores the instance ID's in the `ObjectInst` type. The instance-ID of the filter is determined by the last input argument `inst`, which is automatically incremented in a filter function call.

First/second-order low-pass filter: first- and second-order low-pass filters pass signals with frequencies lower than the cross-over frequency, but attenuate signal components above this frequency. This filters are represented by

$$\mathcal{L}_1(s) = \frac{\omega_\mathrm{c}}{s + \omega_\mathrm{c}}, \qquad \mathcal{L}_2(s) = \frac{\omega_\mathrm{c}^2}{s^2 + 2\zeta\omega_\mathrm{c} s + \omega_\mathrm{c}^2} \tag{2}$$

where $\omega_\mathrm{c} = 1/\tau_\mathrm{c}$ represents the corner frequency, and ζ is the damping coefficient.

First-order high-pass filter: a first-order high-pass filter passes signals with frequencies higher than the cut-in frequency, but attenuates signal components below this frequency. This filter is represented by

$$\mathcal{H}_1(s) = \frac{s}{s + \omega_\mathrm{c}}. \tag{3}$$

Notch filter: a notch filter, often also referred to as a band-stop or a band-rejection filter, passes most of the frequencies but attenuates in a very specific predefined interval. The filter is represented by

$$\mathcal{N} = \frac{s^2 + 2\beta_1\omega_\mathrm{n} s + \omega_\mathrm{n}^2}{s^2 + 2\beta_2\omega_\mathrm{n} s + \omega_\mathrm{n}^2}, \tag{4}$$

where the magnitude and ratio between β_1 and β_2 determines the amplification/attenuation magnitude and width, respectively.

Inverted-notch filter, with decreasing slopes: an inverted notch with decreasing slopes on both sides of ω_n provides extra attenuation of frequencies outside the filtering frequency, for both higher and lower frequencies. The transfer function describing this filter is given as

$$\mathcal{N}_\mathrm{s}(s) = \frac{s}{(Q/\omega_\mathrm{n})s^2 + s + Q\omega_\mathrm{n}}, \tag{5}$$

where Q is the quality factor coefficient: lower and higher values of Q give a broader and sharper selection of the frequency ω_n, respectively.

2.3. Function module

The DRC controller is supplied with frequently used functions which are contained in the function module `Functions.f90`. The included functions are described briefly in this section.

Value/signal saturation: the `saturate()` function saturates a given input signal to a upper and lower value, and returns a real output signal.

Signal rate limiter: a signal rate with respect to time can be bounded by the `ratelimit()` function. The minimum and maximum rate bounds are defined on a per-second basis.

Proportional-Integral (PI) controller: an object-based PI-controller is included by the function `PIController()`. Saturations are included on both the integral action as the output signal to provide anti-integrator wind-up and signal saturation capabilities.

1D-interpolation: a one-dimensional interpolation function `interp1d()` is included to perform evaluation of a value in the interval of a one-dimensional table, of which the x-data should be monotonically increasing.

3. Baseline control overview

This section describes the baseline control implementations and strategies included in the DRC. First, a state-machine determining the state of the operational wind turbine is described in Section 3.1. Then, Section 3.2 describes the baseline pitch and torque controllers. Next, individual pitch control (IPC) and yaw-rate control are outlined in Sections 3.3 and 3.4.

3.1. State-machine

The purpose of the global state-machine is to determine the operational state of the wind turbine, and is included as a subroutine in the function module. Inside this function, separate state-machines are included for pitch and torque control. The operational state is determined by comparing measured turbine quantities and control signal values to controller settings defined in the `ControllerParameters.in` configuration file. Figure 2 gives an overview of the different wind turbine operating regions, indicating internal controller variables and configuration parameters.

3.2. Baseline pitch and torque control

This section presents the pitch and torque control implementations included in the DRC. The controllers use a common generator speed measurement `GenSpeedF` filtered by a second-order low-pass filter with cut-off frequency `CornerFreq`, to calculate the error from the reference parameter. Both controllers act on individual generator speed set points defined by `VS_RefSpd` and `PC_RefSpd` for torque and pitch control.

At the end and beginning of regions 1 and 2.5, two PI torque controllers are implemented to regulate the rotor speed towards the optimal below-rated torque path, and to above-rated operating conditions. The controller gains are defined by `VS_KP` and `VS_KI`. The controllers continuously calculate the below- and above-rated torque references `GenBrTq` and `GenArTq`, and act on different error signals `VS_SpdErrBr` and `VS_SpdErrAr`. As shown in Figure 2, the

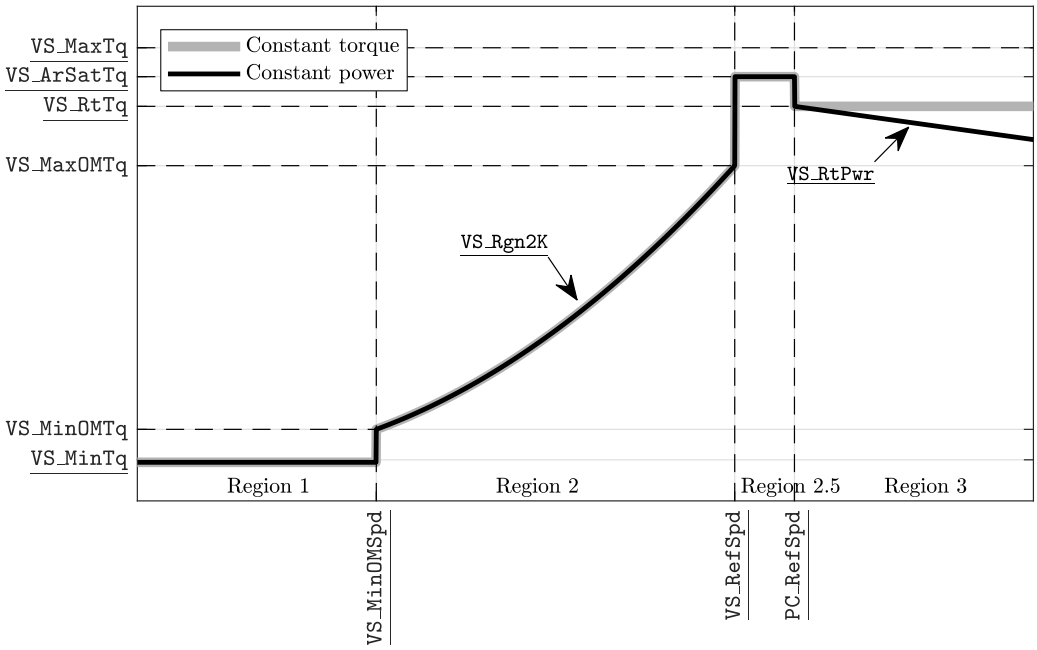

Figure 2. Torque control strategies implemented in the DRC. All variables regarding torque control are indicated by their respective names present in the control parameter file. For above-rated operation (Region 3), the control strategy can be configured to be either in constant torque of constant power mode by the parameter VS_ControlMode.

maximum torque controller saturation VS_ArSatTq can be set independently to provide in less frequent switches between torque and pitch control. The torque signal change rate is limited by VS_MaxRat, and the physical minimum and maximum torque constraints are defined by VS_MinTq and VS_MaxTq. The cut-in speed for below-rated torque control (Region 2) is defined by VS_MinOMSpd, and the optimal mode-gain for tracking $C_{\mathrm{p,max}}$ is set by VS_Rgn2K.

For power regulation during above-rated operation the torque controller can be configured to either deliver a constant torque signal, or actively change the torque signal to obtain a constant power output by setting the parameter VS_ControlMode to 0 or 1, respectively. When constant torque control is selected, the generator torque is VS_RtTq; for constant power tracking, the torque signal varies subject to the instantaneous generator speed and the electrical power set point defined by VS_RtPwr. The generator efficiency VS_GenEff must be defined and should match the value specified in the turbine drivetrain model parameters.

The baseline pitch controller is implemented as a gain-scheduled PI-controller. The proportional and integral gains PC_GS_KP and PC_GS_KI, are scheduled on the commanded pitch angle of the previous controller iteration. The gain information is defined in the ControllerParameters.in parameter file as function of pitch angles PC_GS_angles, and is interpolated in each time step by the interp1d() function.

Pitch control is active in all operating regions. However, based on the turbine operational state PC_State determined by the state machine, the upper saturation bound is limited to the fine-pitch angle PC_FinePit or the maximum physical pitch angle PC_MaxPit. The pitch control signal rate is limited by a lower and upper bound PC_MinRat and PC_MaxRat. The PC_Switch parameter is added to the fine-pitch angle PC_FinePit parameter, and the summed value is used to indicate when the torque controller should switch to region 3.

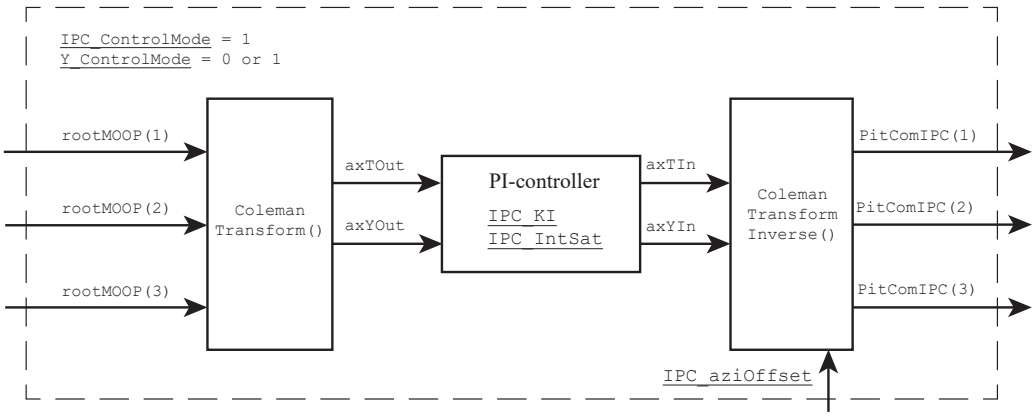

Figure 3. Individual pitch control for blade fatigue load reductions. The out-of-plane blade root moments are transformed in a tilt- and yaw-axis by a 1P Coleman transformation. After PI-control, the resulting pitch angles are transformed back by an inverse Coleman transformation to obtain IPC pitch signals to mitigate 1P fatigue loadings.

3.3. Individual pitch control

An individual pitch control (IPC) implementation based on the Coleman transformation [16] is included in the DRC for two distinct purposes: for fatigue load reductions and blade moment yaw-control (yaw-by-IPC). The two IPC modes cannot be activated simultaneously. Both implementations are described in this section.

IPC for fatigue load reductions: a schematic overview of IPC for blade moment fatigue load reductions is presented in Figure 3. Setting `IPC_ControlMode` to `1` enables IPC to reduce fatigue out-of-plane blade root moments, by adding contributions to the pitch control signals. The implementation focuses on attenuation of the 1P blade load harmonic. The measured blade root out-of-plane moments, together with the rotor azimuth angle, are taken as input to the forward Coleman transformation, resulting in non-rotating rotor tilt- and yaw-moments. Subsequently, the two signals are integrated with a gain `IPC_KI`, and the integrated signal is saturated to `IPC_IntSat` to prevent excessive pitch contributions. Finally, the two pitch signals in the non-rotating frame are subject to a reverse Coleman transformation to obtain actual

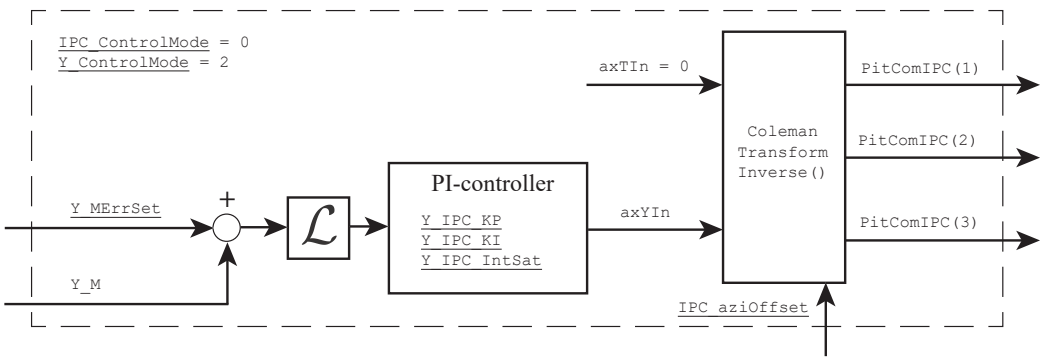

Figure 4. A IPC yaw control implementation for a wind turbine where the nacelle is mounted on the tower, in a free-damped fashion. The transformed tilt pitch angle is nullified, and the yaw angle is actively controlled by the error between the set point and measured yaw misalignment.

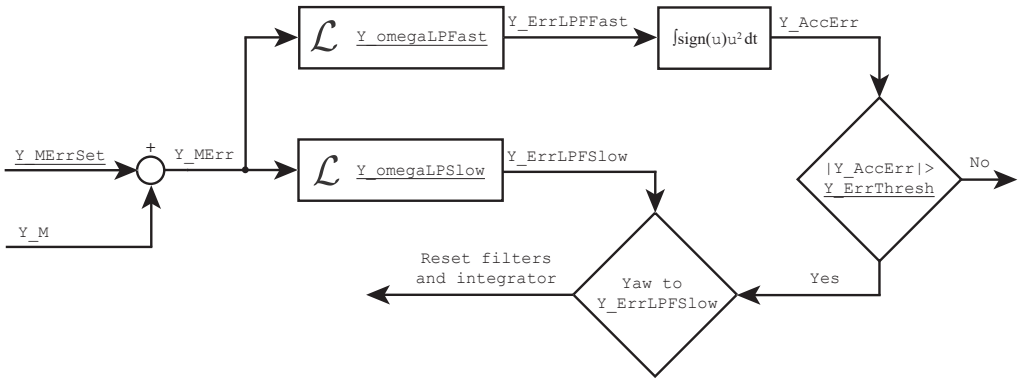

Figure 5. Yaw rate control uses the error between the misalignment set point and the measured misalignment with the dominating wind direction to intermittently perform yaw manoeuvres.

implementable pitch signals. A phase offset IPC_aziOffset can be added to the azimuth angle in the reverse transformation.

Yaw-by-IPC: IPC can be configured to act in a yaw-by-IPC set-up, by setting Y_ControlMode to 2. By doing so, the blades induce a yaw moment on the entire rotor to actively regulate or track a yaw misalignment set point. With yaw-by-IPC, a 1P contribution is added to the pitch signals resulting in a yaw moment over the entire rotor. Normally, this type of control is present in downwind wind turbines, where the nacelle is mounted in a free-damped fashion on the tower support structure [9, 15]. A schematic overview of the implementation is presented in Figure 4. In this figure it is shown that the yaw misalignment error Y_MErr, which is the difference between measured yaw misalignment Y_M and set point Y_MErrSet, is fed to a PI-controller with proportional and integral gains Y_IPC_KP and Y_IPC_KI. The control output signal is saturated by Y_IPC_IntSat.

3.4. Yaw-rate control

The yaw-rate controller uses measurements from a wind vane located downwind, i.e., seen from upwind the vane is positioned behind the rotor and tower. The wind vane measures the nacelle yaw-misalignment with respect to the dominating wind direction, but does not give information on the absolute nacelle orientation. Yaw motors with a fixed yaw-rate are used for yaw movements. The yaw-rate control implementation does not provide continuous alignment, but intermittently aligns the turbine nacelle when a predefined threshold is exceeded.

The implementation adapted from [8] and schematically depicted in Figure 5, is slightly adjusted to allow for yaw-angle offsets. Two distinct low-pass filters on the yaw-error signal Y_MErr are employed. A fast low-pass filter with a higher corner-frequency Y_omegaLPFast is used to determine when the turbine should yaw according to some predefined threshold Y_ErrThresh. The squared value of the yaw-error is used to obtain a higher penalization for larger yaw excursions, and is integrated into Y_AccErr. Once the integrated error exceeds the predefined threshold, the value from the low-pass filter with a lower corner-frequency Y_omegaLPSlow is taken to determine the yaw time using the fixed angular yaw speed parameter Y_Rate.

4. Results

This section presents the performance and behavior of the DRC baseline controller in combination with OpenFAST v1.0.0-x64. The DTU10MW wind turbine model introduced in Section 1, is used for evaluation purposes in two distinct simulations.

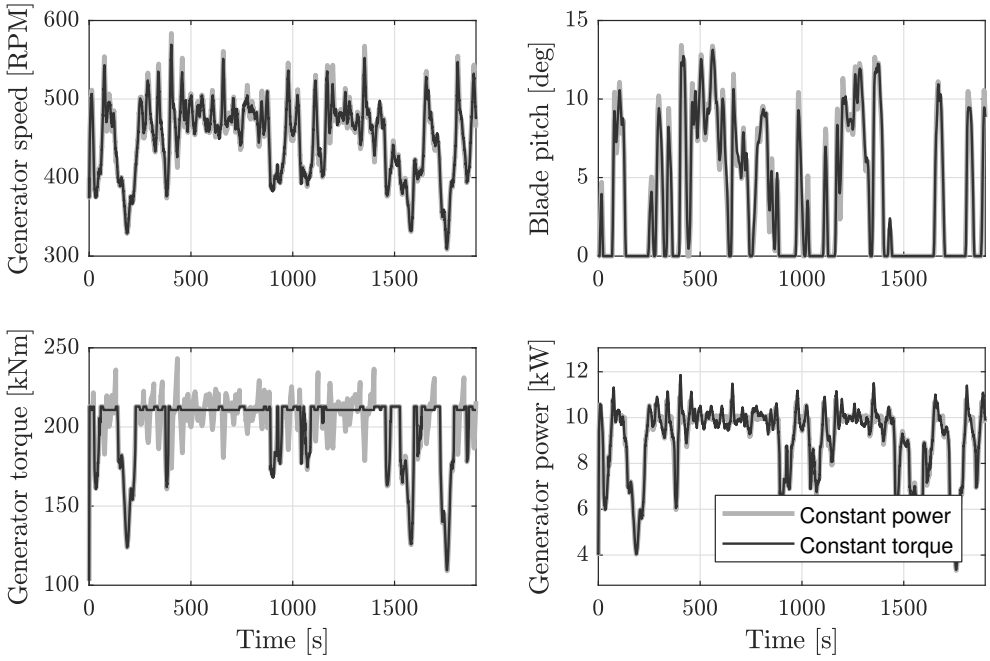

Figure 6. Simulation of the DTU10MW with constant power and constant torque control, subject to a wind profile with mean total velocity of 12 m/s and IEC turbulence characteristic and type NTM-1A.

First, the wind turbine is subject to a IEC-1A Normal Turbulence Model (NTM) wind profile with a mean wind speed of 12 m/s. Using this wind profile, the torque controller of the DRC is configured in constant power and constant torque mode, and the results are presented in Figure 6. It is shown that for both cases the controller handles below- and above-rated operation well by switching between torque and pitch control. Power regulation is improved using the constant power control strategy, at cost of continuous adjustment of the generator torque. The effect on pitch control and the generator speed for the two strategies is negligible.

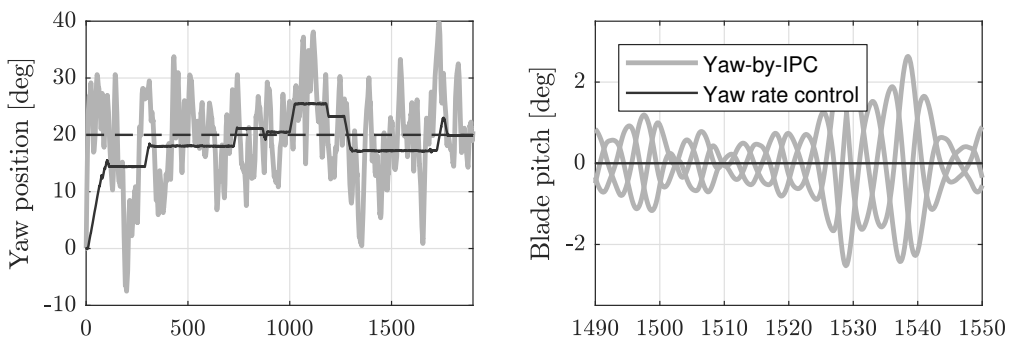

Figure 7. Simulation of the yaw-rate and yaw-by-IPC control implementation, tracking a constant yaw-misalignment set point of 20 deg. The yaw-rate controller is only active intermittently, while the yaw-by-IPC controller continuously regulates the free-yaw turbine around the misalignment set point using distinct blade pitch angles. For the latter case, the blade pitch rate does not exceed 3 deg/s.

Secondly, a simulation is performed to showcase the performance of the yaw-rate and yaw-by-IPC controllers by tracking a misalignment set point of 20 deg perpendicular to the rotor plane. Results are shown in Figure 7, where the same wind profile as defined in the first simulation is used. For yaw-by-IPC control, the yaw spring-stiffness of the DTU-10MW model is removed while retaining the yaw-damping between nacelle and tower.

Conclusion and outlook

The DRC provides an open and community-driven wind turbine baseline controller, and aims in being the reference controller for evaluation of new control algorithms. The controller is set up in such a way that it is applicable to each wind turbine model in frequently used wind turbine simulation software. Only a single parameter file is needed to configure the controller, which abandons the need for recompilation under a change in controller settings. Because of the modular set-up, the existing baseline control implementations are easily replaced, which enables for convenient comparison, evaluation and reproducibility of new algorithms. Although the baseline controller is build on basic PI controllers, more advanced control methods can be implemented by incorporating external Fortran libraries. The next step is to incorporate a general applicable wind speed estimator, enabling closed-loop tip-speed ratio tracking during below-rated operation. In later stages, the DRC functionality will be further extended enabling a distributed control set-up in a wind farm setting.

Acknowledgments

We thank J. Govers and J. Hoorneman (TU Delft) for their dedication initiating the DRC project. We acknowledge CL-Windcon for their contributions improving the code.

References

[1] Jonkman J, Butterfield S, Musial W and Scott G 2009 *Definition of a 5-MW reference wind turbine for offshore system development* Tech rep. NREL, Golden, CO.
[2] Bak C, Zahle F, Bitsche R, Kim T, Yde A, Henriksen L, Andersen P, Natarajan A and Hansen M 2013 *Design and performance of a 10MW wind turbine* H, J. Wind Energy
[3] DNV-GL 2018 Bladed; `https://www.dnvgl.com/energy`
[4] NWTC 2018 OpenFAST `http://openfast.readthedocs.io`
[5] Hansen MH and Henriksen LC 2013 *Basic DTU wind energy controller* DTU Wind Energy. DTU Wind Energy E, no. 0028
[6] Garrad Hassan & Partners Ltd *Bladed User Manual version 4.2*
[7] Lahey T M and Ellis T 1994 *Fortran 90 programming* (Addison-Wesley Longman Publishing Co., Inc.)
[8] Kragh K and Fleming P 2012 Rotor Speed Dependent Yaw Control of Wind Turbines Based on Empirical Data **AIAA 2012** 1 - 9
[9] Van Solingen E, Beerens J, Mulders S, De Breuker R and Van Wingerden JW 2016 *Control design for a two-bladed downwind teeterless damped free-yaw wind turbine* Mechatronics **36** 7796
[10] Boersma S, Doekemeijer BM, Gebraad PMO, Fleming PA, Annoni J, Scholbrock AK, Frederik JA and Van Wingerden JW 2017 *A tutorial on control-oriented modeling and control of wind farms* IEEE **ACC 2017** 1 - 18
[11] Fleming P, et al. 2013 *SOWFA Super-Controller: A High-Fidelity Tool for Evaluating Wind Plant Control Approaches* (No. NREL/CP-5000-57175). NREL, Golden, CO
[12] Project G 2018 GNU Fortran (GFortran) `https://gcc.gnu.org/fortran/`
[13] Intel 2018 Intel Fortran Compiler `https://software.intel.com/en-us/fortran-compilers`
[14] Oppenheim A V 1999 *Discrete-time signal processing* (Pearson Education India)
[15] Schorbach V and Dalhof P 2012 *Two bladed wind turbines: antiquated or supposed to be resurrected* Proceedings of the EWEA Conference
[16] Bir G 2008 *Multi-blade coordinate transformation and its application to wind turbine analysis* **AIAA** 46th