nrel$_5 mw_c ontroller Documentation$

Release 0.0.post0.dev2+g2962119

Rick Lupton

Feb 21, 2019

Contents

1 Contents **3**
- 1.1 License . . . 3
- 1.2 Contributors . . . 3
- 1.3 Changelog . . . 3
- 1.4 nrel_5mw_controller . . . 4

2 Indices and tables **9**

Python Module Index **11**

This is the documentation of **nrel_5mw_controller**, an implementation of the NREL 5MW wind turbine controller in Python. It was originally written for Rick Lupton's PhD thesis and this paper.

CHAPTER 1

Contents

1.1 License

1.2 Contributors

- Rick Lupton <mail@ricklupton.name>

1.3 Changelog

1.3.1 Version 0.1

- Feature A added

- FIX: nasty bug #1729 fixed
- add your changes here!

1.4 nrel_5mw_controller

1.4.1 nrel_5mw_controller package

Submodules

nrel_5mw_controller.combined_controller module

Combined controller, including both torque and pitch control.

class nrel_5mw_controller.combined_controller.**CombinedController**(*torque_params*, *pitch_params*, *torque_timestep*, *pitch_timestep=None*, *const_power_min_pitch=0*)

Bases: object

Controller combining both torque and pitch control.

Parameters

- **torque_params** (*dict*) – passed to the TorqueController
- **pitch_params** (*dict*) – passed to the PitchController
- **torque_timestep** (*float*) – timestep for the torque controller
- **pitch_timestep** (*float, optional*) – timestep for the pitch controller. Defaults to the same as torque_timestep.
- **const_power_min_pitch** (*float, optional*) – The minimum pitch angle to start forcing constant power mode for the torque controller. Default 0.

classmethod from_yaml(*filename*)
Read controller params from 'controller' section of YAML file

pitch_demand
The current pitch demand from the pitch controller.

step(*time*, *measured_speed*, *measured_pitch*)
Step both controllers forwards in time.

This is the main method of the controller class.

Parameters

- **time** (*float*) – the current timestamp
- **measured_speed** (*float*) – current measured generator speed
- **measured_pitch** (*float*) – current measured pitch angle

torque_demand
The current torque demand from the torque controller.

nrel_5mw_controller.pitch_controller module

Implementation of the pitch controller for the NREL 5MW wind turbine controller.

class nrel_5mw_controller.pitch_controller.**PitchController**(*timestep*, *params*)
Bases: object

Time-stepping pitch controller for the NREL 5MW wind turbine.

It expects the following parameters:

- `proportional gain`: Proportional gain of the PI controller
- `integral gain`: Integral gain of the PI controller
- `pitch schedule doubled angle`: This is the angle at which the pitch controller gain is halved.
- `pitch angle min`: Minimum pitch angle limit
- `pitch angle max`: Maximum pitch angle limit
- `pitch rate limit`: Maximum pitch rate limit (up or down)
- `rated speed`: The generator speed setpoint for the controller
- `speed filter corner freq`: The frequency of the generator speed filter.

classmethod from_yaml(*filename*)
Read controller params from 'controller' section of YAML file

get_pitch_demand(*speed_error*, *speed_error_int*, *GK*)

get_scheduled_gain(*pitch*)
Calculate the gain schedule factor.

initialise(*time*, *measured_speed*, *measured_pitch*)
Initialise the controller.

Parameters

- **time** (*float*) – current timestamp
- **measured_speed** (*float*) – current measured generator speed
- **measured_pitch** (*float*) – current measured pitch angle

reset()
Reset the controller state.

step(*time*, *measured_speed*, *measured_pitch*)
Step the controller forwards to the next timestep.

This is the main method of the controller class.

Parameters

- **time** (*float*) – the current timestamp
- **measured_speed** (*float*) – current measured generator speed
- **measured_pitch** (*float*) – current measured pitch angle

nrel_5mw_controller.torque_controller module

Implementation of the NREL 5MW wind turbine torque controller.

class nrel_5mw_controller.torque_controller.**TorqueController**(*timestep*, *params*)
Bases: object

Time-stepping torque controller for the NREL 5MW wind turbine.

It expects the following parameters:

- `rated speed`: The generator speed setpoint for the controller
- `rated power`: The generator power setpoint for the controller, for constant power mode
- `slip percent`: Generator slip rate, to calculate the synchronous speed
- `opt constant`: the k coefficient for the optimal speed control region
- `speed filter corner freq`: The frequency of the generator speed filter.
- `cut in speed`: cut in generator speed
- `opt min speed`: minimum generator speed for optimal control (linear ramp between cut in speed and this speed)
- `torque max`: maximum generator torque
- `torque rate limit`: Maximum torque rate limit (up or down)

Optional parameters:

- `constant torque`: control for this constant torque above rated, instead of constant power.

classmethod from_yaml(*filename*)
Read controller params from 'controller' section of YAML file

get_torque(*spd*, *const_power*)

initialise(*time*, *measured_speed*)
Initialise the controller.

Parameters

- **time** (*float*) – current timestamp
- **measured_speed** (*float*) – current measured generator speed

reset()
Reset the controller state.

step(*time*, *measured_speed*, *force_constant_power*)
Step the controller forwards to the next timestep.

This is the main method of the controller class.

Parameters

- **time** (*float*) – the current timestamp
- **measured_speed** (*float*) – current measured generator speed
- **force_constant_power** (*bool*) – force constant power mode?

nrel_5mw_controller.util module

Utility functions.

nrel_5mw_controller.util.**saturate**(*x*, *a*, *b*)

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

nrel_5mw_controller, 7
nrel_5mw_controller.combined_controller, 4
nrel_5mw_controller.pitch_controller, 5
nrel_5mw_controller.torque_controller, 6
nrel_5mw_controller.util, 7

Index

C

CombinedController (class in nrel_5mw_controller.combined_controller), 4

F

from_yaml() (nrel_5mw_controller.combined_controller.CombinedController class method), 4
from_yaml() (nrel_5mw_controller.pitch_controller.PitchController class method), 5
from_yaml() (nrel_5mw_controller.torque_controller.TorqueController class method), 6

G

get_pitch_demand() (nrel_5mw_controller.pitch_controller.PitchController method), 5
get_scheduled_gain() (nrel_5mw_controller.pitch_controller.PitchController method), 5
get_torque() (nrel_5mw_controller.torque_controller.TorqueController method), 6

I

initialise() (nrel_5mw_controller.pitch_controller.PitchController method), 5
initialise() (nrel_5mw_controller.torque_controller.TorqueController method), 6

N

nrel_5mw_controller (module), 7
nrel_5mw_controller.combined_controller (module), 4
nrel_5mw_controller.pitch_controller (module), 5
nrel_5mw_controller.torque_controller (module), 6
nrel_5mw_controller.util (module), 7

P

pitch_demand (nrel_5mw_controller.combined_controller.CombinedController attribute), 4
PitchController (class in nrel_5mw_controller.pitch_controller), 5

R

reset() (nrel_5mw_controller.pitch_controller.PitchController method), 5
reset() (nrel_5mw_controller.torque_controller.TorqueController method), 6

S

saturate() (in module nrel_5mw_controller.util), 7
step() (nrel_5mw_controller.combined_controller.CombinedController method), 4
step() (nrel_5mw_controller.pitch_controller.PitchController method), 5
step() (nrel_5mw_controller.torque_controller.TorqueController method), 6

T

torque_demand (nrel_5mw_controller.combined_controller.CombinedController attribute), 4
TorqueController (class in nrel_5mw_controller.torque_controller), 6