Process Control Project

Summer semester 2023

Pascal Bock
Group: ET_STK1

August 15, 2023

Contents

1. Process description, problem definition and modelling

In this section the process of a wind turbine generating electricity is examined. Fragoso et al. analyse a lab-scale model of wind turbine with regard to decoupling control and multivariable robust control [5]. The section starts with a brief introduction of the process, then analyses the control design aspects and summarizes with the model equations and further reading material.

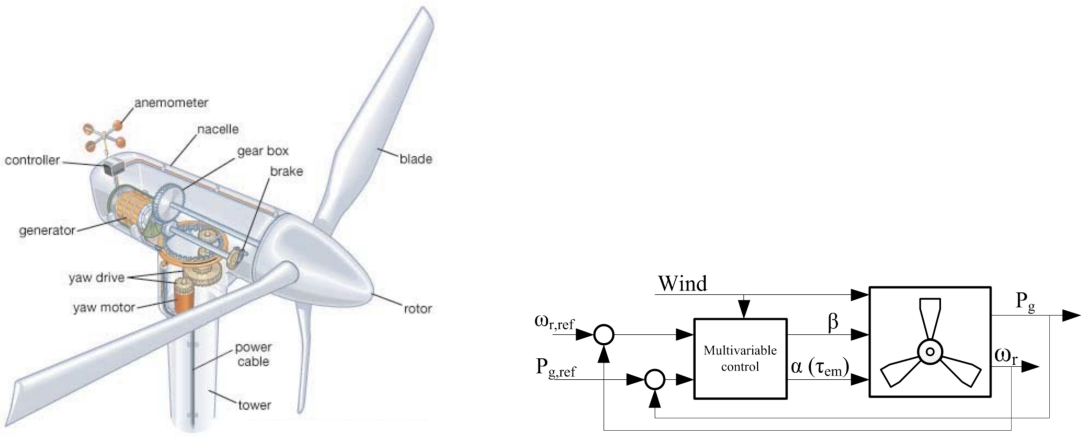
1.1. Introduction and process description

According to the International Energy Agency (IEA), the renewable electricity generation worldwide by wind has gone up from 342 205.0 GW h in 2010 to 1 598 080.0 GW h in 2020 [2]. Looking at Germany the same trend can be seen by IEA: From 38 547.0 GW in 2010 to 113 848.0 GW h in 2020. By comparing this with solar PV (11 729.0 GW h in 2010 to 49 992.0 GW h in Germany), the huge increase becomes clear [2].

The system by Fragoso et al. consists of a lab-sized rotor with two blades and a permanent magnet DC electric generator placed inside a wind tunnel [5]. This system is decomposed into two subsystems: First the mechanical subsystem describing the rotor and second the electrical subsystem describing the generator. The system equations for both subsystems are derived and a linear model is developed. To better understand the coupling of system states the relative gain array is analysed. In the second half, different tools for controller design are discussed and later simulation data is compared with experimental data.

For defining a control objective it is necessary to know the four operating condition with respect to wind speed: (1) cut-in, (2) partial load, (3) transition, (4) full load and (5) cut-out. Partial and full load are the two status of most interest. Transition describes the pass from operating state (2) to (4). While cut-in and cut-out are the state with too little and too much wind to operate the turbine.

1.2. Control design aspects



(a) Process diagram of a windturbine [6].

(b) Control block diagram of a a wind turbine model [5]. Consisting of the input variables (left), the output variables (right), the controller (Multivariable control), the wind turbine itself and the wind as a disturbance.

Figure 1: Comparison of a process diagram and the control block representation of a wind turbine.

Control objective Fragoso et al. discusing two different objective at two different operating states. If the turbine is operated in partial load the objective is to maximize the power that can be captured from the wind. The objective changes to maintaining the power constant at the rated power if the turbine is operated in full load.

Input variables Pitch angle β and generator torque τ_{em} are commonly used as input variables.

Output variables The rotational speed ω_r and generated electric power P_G are the output variables of the system.

Constraints For controller design a linear model (derived by identification) is obtained by the authors. For further constraints Figure 13 and Figure 14 in Fragoso et al. publications are analyzed. The following constraints can be seen in the output and control signals.

Table 1: Possible constraints for input and output variables.

	ω_r [rpm]	P_G [W]	β [°]	α [%]
Min	1550	5	0	60
Max	1750	7	25	75

Operating characteristics The process is a continuous batch process. The wind mill produces a laminar flow of air. This is not the case in general. If the stream of air is not laminar turbulences will occur at the blades. This will introduces further disturbances which hard to calculate or estimate.

Control structure The rotational speed ω_R and generated electric power P_G are the controlled variables, blade pitch angle β and the duty cycle α are the manipulated variables and the wind speed v is counted as a disturbance. The system is a multiple input multiple output (MIMO) system or two input and two output (TITO) system. In total one H_∞ based robust controller and three decoupling approaches are developed.

1.3. Model equation

Fragoso et al. defining their model through nonlinear ordinary differential equations. They distinguish between the electromechanical and electrical subsystems. Here only a brief overview is given. All relevant equations and parameter are published [5].

For the mechanical subsystem the rotational speed ω_R is given by

$$J_t \frac{d\omega_r}{dt} = \tau_a - \tau_{em} \tag{1}$$

Here J_t is the total inertia momentum of the system. τ_{em} is the electromechanical torque and τ_a the aerodynamic torque which can be expressed as a nonlinear function of wind speed v and a power coefficient $C_p(\lambda, \beta)$. Where λ is the tip speed ratio and β the pitch angle. Fragoso et al. especially describe the power coefficient as an important parameter for the control and emphasises its nonlinear influence to the systems dynamics.

The electromechanical torque and the generated electrical power define the electrical subsystem.

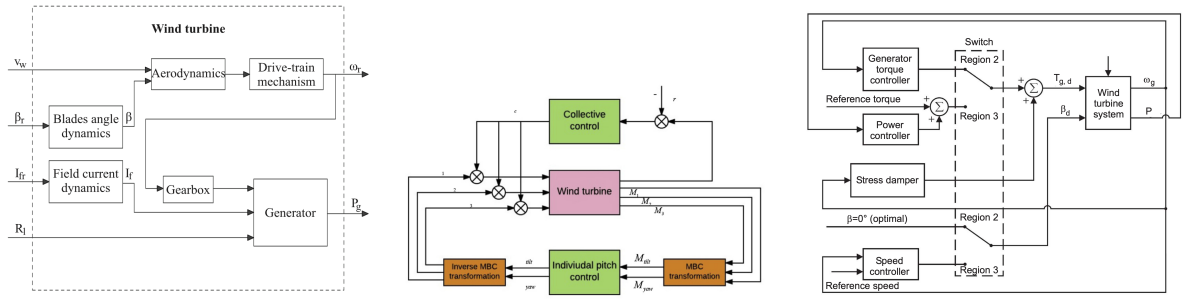
$$\tau_{em} = k_t i_g \tag{2}$$

$$P_g = \eta \tau_{em} \omega_r \tag{3}$$

In both equations the efficiency is represented by k_t and η respectively for the electromechanical torque and the generator.

1.4. Further reading

Adánez et al. published a control approach for a wind turbine based on incremental state model in 2018 [1]. The authors gave an overview over recent literature before describing their model of a wind turbine. In comparison to Fragoso et al. their schematics showing a more detailed look at the different parts of the system (Figure 2a). At the same time, however, it can be seen that very similar input and output



(a) Process block diagram of a wind turbine with dynamics of field current and blade angles [1].

(b) The control scheme used by Yuan, Cheng and Tang [7].

(c) Visualization of switching control structures when wind turbine is in different operating conditions [6].

Figure 2: Comparison of different visualization of control and system structures.

variables are used. They develope a multivariable discrete state model and an incremental state model. Besides that, an observer for the state feedback controller is derived.

Yuan, Cheng and Tang are describing in their work from 2020 the problems arising from the increasing blade sizes of modern wind turbines and thereby developing challenges in controller design [7]. In contrast to the aforementioned publication the authors do not show a scheme of the overall system, but instead providing a visual representation of the control structure (Figure 2b). The system itself is implemented in third party code and not as clearly shown as in the other publications.

Simani gives an full overview of the topic of modeling wind turbines [6]. He summarizing not only models for the generator and blades but also for the wind turbine tower itself and the dynamics regarding blade motions. In contrast to all other mentioned publication he is giving an overview of switching control structure when the turbine is in different operating conditions (Figure 2c). Also he is emphasising the switch in control objectives in different operating conditions.

1.5. Summary

The work of Fragoso et al. seems to fit best in the criteria of the course by clearly naming all important design aspects and deriving comprehensible and probably easy to re-implement model.

1.5.1. Simulating the model

In Table 2 Fragoso et al. published several transfer function for different wind speed. These transfer function could be used to simulate the model.

For the ordinary differential equation all the necessary parameters for a simulation seem to be given. Also initial condition are formulated or could be read from Figure 13 and 14.

2. System Analysis

This section deals with the analysis of the aforementioned system of a wind turbine. First, the description of the ODE system, which is insufficient for an implementation, is presented (subsection 2.1) and the problems encountered are briefly described.

Second, the implementation of the system via transfer function is introduced subsection 2.2. All important steps to implement the system in Matlab are shown. By using figures published by Fragoso et al.–[5] the implementation in Matlab is verified.

The following chapter (section 3) is dealing with controller design. One important step of controller design is the relative gain array, which is conducted at the end of tis section.

2.1. Simulation of ODE system

Fragoso et al. [5] present a system of ODE for the wind turbine model. As described in subsection 1.3 the model is subdivided into two parts. The electrical subsystem is based in on two algebraic equations (Equation 2, Equation 3).

For the mechanical part an ODE is given

$$J_t \frac{d\omega_t}{dt} = \tau_a - \tau_{em}, \tag{4}$$

where τ_a is the aerodynamic torque and defined as

$$\tau_a = \frac{1}{2}\rho\pi R^3 \frac{C_p\left(\lambda,\beta\right)}{\lambda} v^2. \tag{5}$$

The power coefficient C_p is only described by a figure (see Figure 4 in [5]). In Fragosos work [4, p. 54 ff.] a function for C_p is derived and also a figure shows the global course of the function (see Figure 3.8, p. 56). In this function nine heuristic parameters are used to determine the characteristics. It is unclear if this function is also used in the work of Fragoso et al. or not.

In the appendix of Fragosos PhD-thesis there are tabulated value for C_p. These were implemented as a lookuptable in Matlab. By using the lookup table, values for C_p could be determined, but the solutions of the ODE were not in a realistic range of values.

```
% Loading tabulated values as .csv file
p.lookup = readtable('lookuptable_c_p.csv');
% Setting corresponding column names
p.lookup.Properties.VariableNames = {'wind_speed','rotor_speed','U','
    i_a','P_e','P_m','lambda','C_p','C_q','beta'};

% Looking up values for values of lambda and beta
function C_p = power_coefficient(lambda,beta,lookup)
    C_p = interp1(lookup.lambda(lookup.beta==beta),lookup.C_p(lookup.
       beta==beta),lambda);
end
```

Listing 1: Matlab function for lookuptable. With this function we can get a corresponding C_p value for given β and λ. The `.csv`-file can be found in the GitHub repository under

Even if it is possible to get correct values for C_p, we still have only one ODE and a algebraic equation. This is not representing a systems with two systems states.

Due to the problems mentioned here, the representation and solution of the system in state space was not pursued further and the transfer function mentioned in the Fragose et al. was used instead.

2.2. Implementation of transfer function model

All controllers will be designed based on a linear model. For different wind speeds v Fragoso et al. identified five different linear models. The pitch angel β and the duty cycle α are the systems states. The outputs are the rotational speed of the blade ω_r and the generated electric power P_g.

$$\begin{pmatrix} \omega_r \\ P_g \end{pmatrix} = G(s) \begin{pmatrix} \beta \\ \alpha \end{pmatrix} + G_D(s)v \tag{6}$$

The transferfunctions G and G_D where identified for windspeed $v = [6, 7, 8, 9, 10]\,\mathrm{m\,s^{-1}}$. The results can be found in [5, Table 2] and will not be reprinted here.

Table 2: Compilation of the most important variables, their meaning, unit and value range. The value ranges in italics are estimated values derived from figures. In addition, only the deviation Δv is given, as we consider the disturbance in the following sections only as a deviation from zero. That is why we also consider a maximum value range between -1 and 1, as we would enter the *domain* of another transfer function for larger values.

Variable	Name	Interpration	Unit	Range
Δv	change in windspeed	disturbance	$\mathrm{m\,s^{-1}}$	$[-1, 1]$
β	pitch angle	state	°	$[0, 25]$
α	duty cycle	state	1	$[0, 1]$
ω_r	rotational speed of blade	input	rpm	$[1400, 2000]$
P_g	generated electric power	input	W	$[3, 9]$

Implementation in Matlab

To generate the transfer functions in Matlab the `tf`-command and the `table` function are used.

```
% Wind speed: 6 m/s
G_S_num_6 = {-8.059,-405.4;-0.0081479,4.4195};
G_S_denom_6 = {[75.27,17.35,1],[28.25, 16.47, 1];[0, 16.873, 1],[0,
    1.7589, 1]};
G_D_num_6 = {364.9; 1.139};
G_D_denom_6 = {[118.3, 21.76, 1];[65.28, 17.09, 1]};

... Wind speeds 7,8,9 an 10 m/s ...

% Making a table to store all transfer_function
varNames = ["wind_speed","G_S_numerator","G_S_denominator","
    G_D_numerator","G_D_denominator","G_S","G_D","RGA"];
table_tf = table;
% Naming the columns in the table
table_tf.Properties.VariableNames = varNames;

% Filling the table with numerator and denominator
table_tf(1,1:5) = {6,G_S_num_6,G_S_denom_6,G_D_num_6,G_D_denom_6};
...
table_tf(5,1:5) = {10,G_S_num_10,G_S_denom_10,G_D_num_10,G_D_denom_10};
table_tf.Properties.VariableNames = varNames(1:5);

% For loop for creating transfer function by using tf-command
for i = 1:5
    table_tf{i,6} = {tf(table_tf{i,"G_S_numerator"}{:},table_tf{i,"
        G_S_denominator"}{:})};
```

```
24      table_tf{i,7} = {tf(table_tf{i,"G_D_numerator"}{:},table_tf{i,"
           G_D_denominator"}{:})};
25      table_tf{i,8} = {NaN};
26  end
```

For each transfer function modeling the system behaviour for different wind speeds we have a numerator and denominator for the systems transfer function G and the disturbance system G_D. Those are stored as arrays `G_num_6`, `G_D_denom_6`, etc. , where the 6 at the end depicting the transfer function for a wind speed of $6\,\mathrm{m\,s^{-1}}$.

In line 17 ff. the numerator and denominator of all transfer function are stored in the table initialized in line 12. Using the `tf`-command inside a `for-loop` (line 23 ff.) the transfer function is defined. In line 14 every column of the table gets a name.

2.2.1. Verifying the implementation

The goal of verification is the recreation of relative step responses shown by Fragoso et al. (Figure 3). The figure is used as a benchmark. If the global course of the step response can be readjusted, then one can assume that the model is implemented correctly.

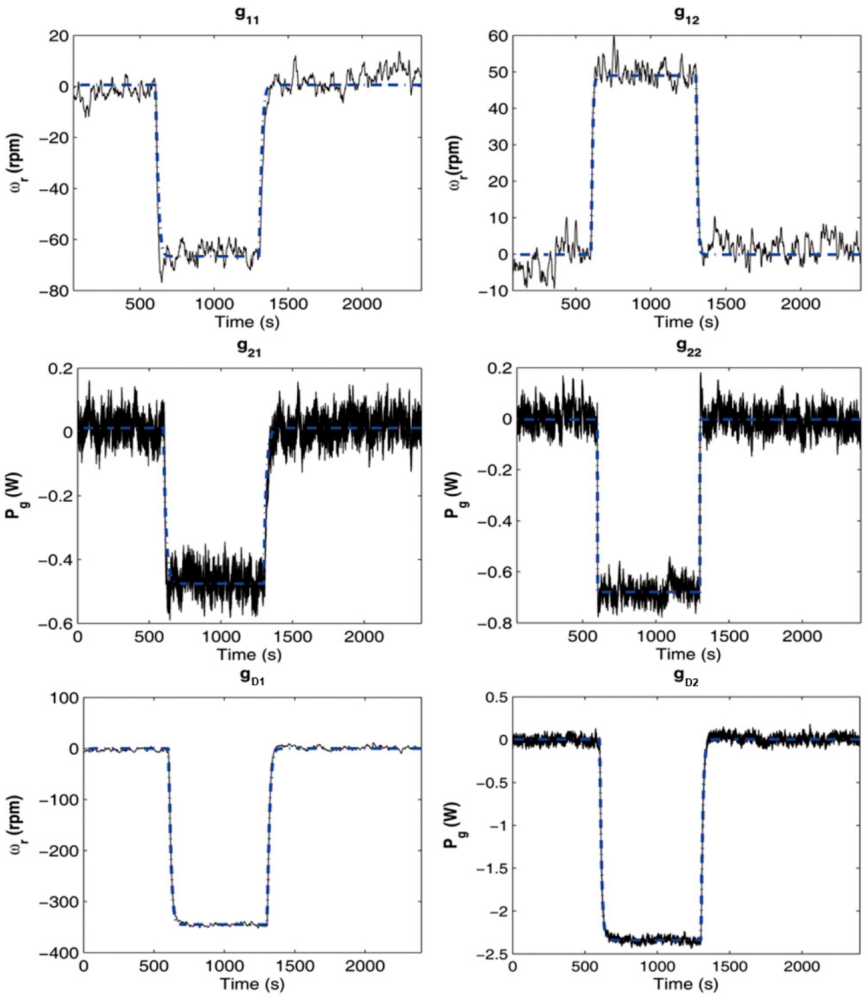


Figure 3: The figure presented by [5] shows the relative step response of the identified system (dashed line) and the real system (solid) line.

The six figures are labeled as g_{ij} as changing input i and showing the influence on output j. The bottom two are referring to the influence of disturbance changes (changes in wind speed) when inputs

are changed. Unfortunately the exact values are not given in the paper and therefore must be reverse engineered.

Table 3: The tables shows the influence of an input to a single output. The table can be seen as an other way of displaying the results from Figure 3.

Figure	Input	Δ	Output	Δ
g_{11}	β	$7°$	ω_r	$-70\,\mathrm{rpm}$
g_{12}	α	-0.1	ω_r	$50\,\mathrm{rpm}$
g_{21}	β	$12°$	P_g	$-0.5\,\mathrm{W}$
g_{22}	α	-0.1	P_g	$-0.7\,\mathrm{W}$
g_{D1}	v_{wind}	$-1\,\mathrm{m\,s^{-1}}$	ω_r	$-300\,\mathrm{rpm}$
g_{D2}	v_{wind}	$-2\,\mathrm{m\,s^{-1}}$	P_g	$-2.3\,\mathrm{W}$

From left to right the first row of Table 3 can be read as: For figure g_{11} the input β is changed by $7°$ and will result in reducing the rotational speed ω_r by 70 rpm. All six figures presented by Fragaso et al. can be recreated by the values given in Table 3. Figure 4 is one example representing the subfigure g_{11} and using the values of the first row.

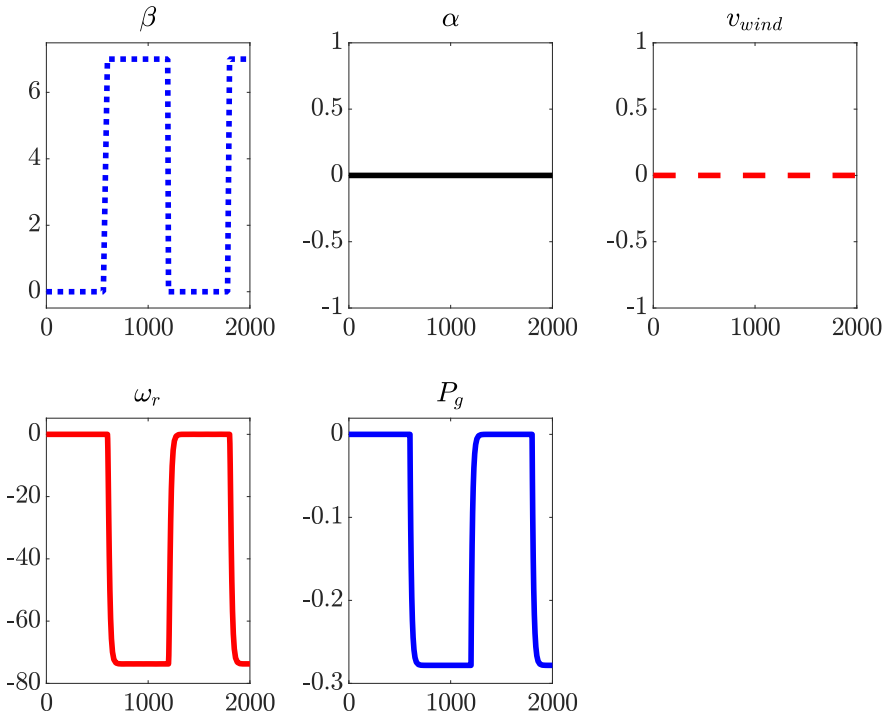


Figure 4: Recreating subfigure g_{11} by plotting inputs, outputs and the disturbance over time. A pulse with a period of 1200 s and a pulse width of 600 s changes β by $7°$. This results in reducing the rotational speed ω_r by 70 rpm.

2.3. Relative Gain Array

The relative gain array is used to obtain a measure of the dependencies of inputs and outputs for a MIMO-system. In this particular example we want to know the influence of rotational speed ω_r and the

electrical power P_g on pitch angle β and duty cycle α.

The RGA-matrix is symmetric and all column- and row-sums a equal to one. For a 2×2-matrix we only need to calculate the element $(1,1)$ and deduce all other values.

$$\Lambda_{RGA} = \begin{pmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{pmatrix}, \text{with} \tag{7}$$

$$\lambda_{12} = \lambda_{21} \tag{8}$$

$$\lambda_{11} + \lambda_{12} = 1 \tag{9}$$

$$\lambda_{21} + \lambda_{22} = 1 \tag{10}$$

Implementation in Matlab

```
1    function RGA = calculate_RGA(G_S_numerator)
2        % Computation of RGA
3        lambda_11 = G_S_numerator{1,1}/(G_S_numerator{1,1} - (
             G_S_numerator{1,2}*G_S_numerator{2,1})/G_S_numerator{2,2} );
4        RGA = [[lambda_11, 1-lambda_11];[1-lambda_11,lambda_11]];
5    end
```

Listing 2: This Matlab function calculates a 2×2-RGA-matrix

In line four the RGA-matrix is calculated. As previously described we only need to calculate λ_{11} and can deduce all other elements of Λ.

Results of RGA

The investigation of the RGA for different wind speeds provides clearly different dependencies between the inputs and states. The results are shown in Figure 5.

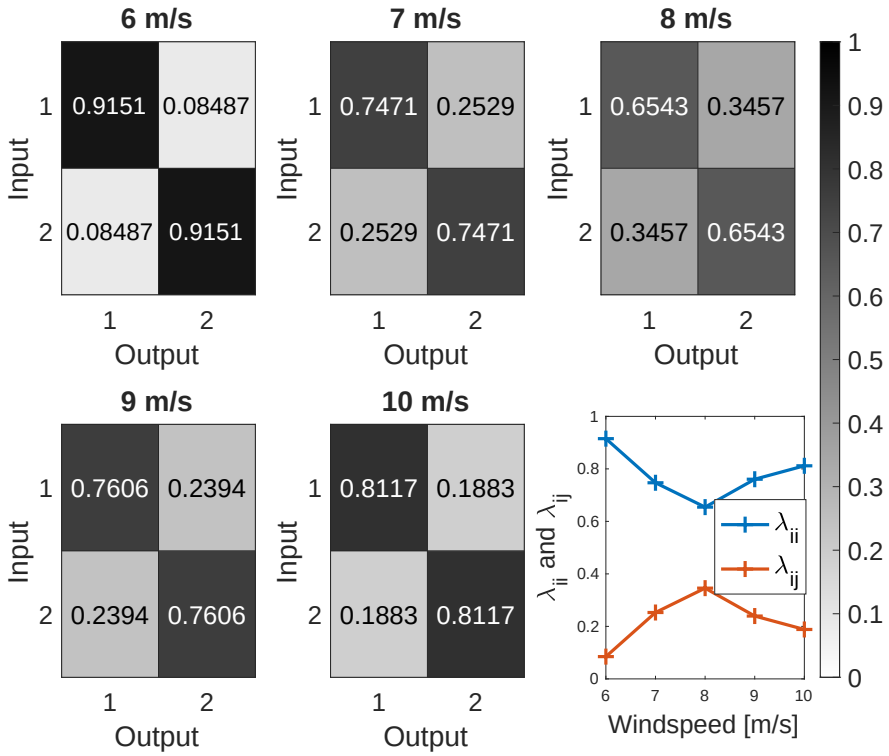


Figure 5: Visualization of the five RGA-matrices and a line plot for λ_{ii} and λ_{ij}.

For very low windspeeds ($6\,\mathrm{m\,s^{-1}}$) the system is nearly decoupled. At a windspeed of $10\,\mathrm{m\,s^{-1}}$, the system can also be considered almost decoupled. If the the windspeed is approaching $8\,\mathrm{m\,s^{-1}}$ the peak coupling is reached. This behaviour can be best seen at the line plot. Both lines almost touch at a windspeed of $8\,\mathrm{m\,s^{-1}}$ and diverge before and after.

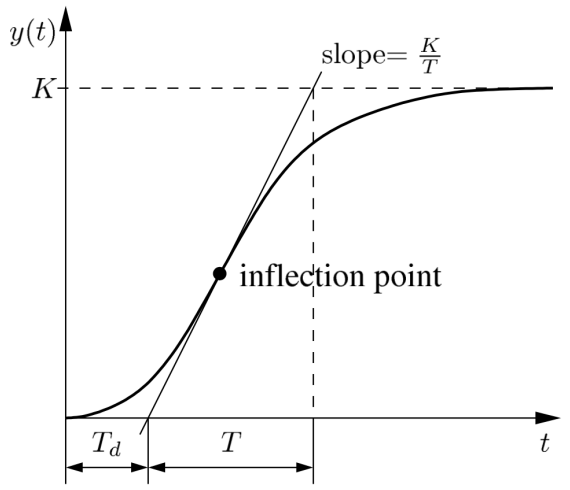
The results achieved are congruent with the findings of Fragaso et al. ([5, p. 7], [4, p. 123])

In the following chapter we will design a controller for the *hardest* condition of a coupled system with windspeeds of $8\,\mathrm{m\,s^{-1}}$.

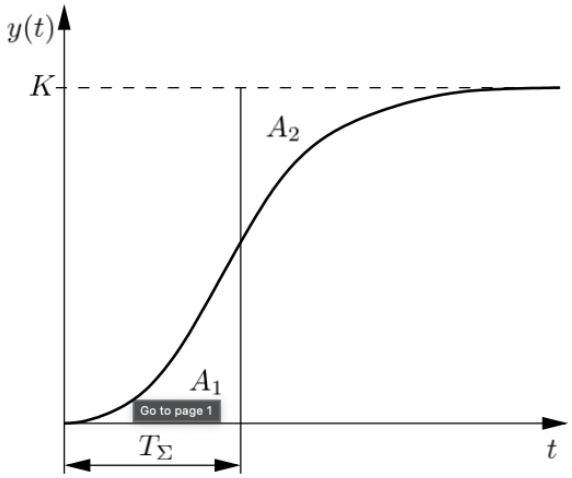
3. Controller Design

In the previous chapters only the open loop behaviour was analyzed. This section focuses on the controller design and closing the loops. Therfore this section describes the implementation of open and closed loop systems in Simulink (subsection 3.3) Three different techniques for controller design are described. First, the Ziegler-Nichols method is shown (subsection 3.1). For this method, two different approximations for the required **f**irst-**o**rder-**p**lus-**t**ime-**d**elay transfer function (FOPTD) are used. The max slope method (subsubsection 3.1.1) uses the inflection point of the step resoponse to determine the control parameters. In contrast, the two-point method (subsubsection 3.1.2) does not use derivatives, but the points in time at which the step response has assumed 28% or 63% of its maximum value. As a result, the method is also considered to be numerically more stable.

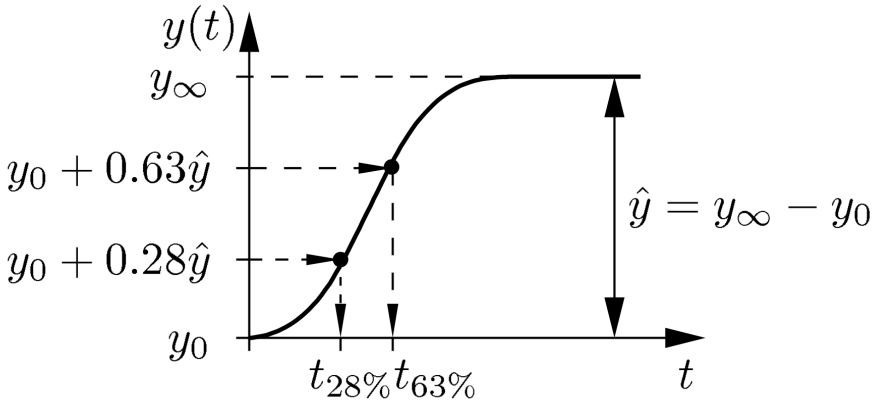
Second, the T-sum method (subsection 3.2) is used to determine control parameters for a PID-controller. Here, the parameters are estimated by using the area under the s-shaped step response.



(a) Max slope approximation[3, Lecture 6, slide 12]



(b) T-sum method [3, Lecture 6, slide 17]



(c) Two-point approximation [3, Lecture 4, slide 6]

Figure 6: Visualization of two different approximation methods for Ziegler-Nichols controller design and the T-sum method for control parameter estimation. All plots from [3]

In the end all methods are discussed and the robustness of some methods is shown (section 4).

3.1. Ziegler-Nichols open loop method

The Ziegler-Nichols open loop method characterises assumes an s-shape step response and an underlying first-order-plus-time-delay transfer function. Looking at the given transfer function by Fragoso et al. ([5,

Table 2]), we do not have this behaviour. Therefore, we need to approximate a FOPTD transfer function. This can be done with two different approaches. First, the max slope approximation is used for controller design and second, the two-point method determines the controller's parameters.

In the following two subsection both methods and their Matlab implementation is discussed.

3.1.1. Max slope approximation

The max slope method focuses around the inflection point of the unit step response.

```
1 [y,t_out] = step(G,t_linspace);
2 [slope, intcpt] = calc_infliction_point_slope(y,t_out);
3 tngt = slope*t_out + intcpt;
4 [K, T, T_d, t_inter] = calc_gain_time_const_time_delay(y,t_out,tngt);
```

Listing 3: This code snippet show the general way of calculating the control parameters with the max slope method.

In line 1 the step response of a transfer function G with the matlab built-in-function `step()` is calculated. `calc_infliction_point_slope()` calculates the infliction point and returns the slope (`slope`) and y-interception (`intcpt`). With these two values the tangent (`tngt`) is calculates in line 3 and in line 4 the variables needed for calculation of the control parameters are returned (`calc_gain_time_const_time_delay()`).

The `calc_infliction_point_slope()`-function calculates the gradient using Matlabs `gradient()`-function and computes slope and y-interception from this.

In `calc_gain_time_const_time_delay()` the values for the maximal gain K, the time delay T_d, the time constant T and the time of interception is computed and returned.

The complete functions can be found in the appendix (subsubsection A.1.1, subsubsection A.1.2).

3.1.2. Two-point approximation

The aforementioned method uses the `gradient()`-function, which is sensitive to noise and outliers. A more robust approach is the two-point method. In Figure 6c we see the points where 28% and 63% of y_∞ is reached. These values are calculated via Matlab and used to compute T, T_d and K.

$$T = \frac{2}{3}\left(t_{63} - t_{28}\right) \tag{11}$$

$$T_d = t_{63} - T \tag{12}$$

$$K = y_\infty \tag{13}$$

In matlab the computation relies on the `interp1()`-function for getting the t-values after calculation of $y_{28,63}$.

```
1  [y,t] = step(G,t_linspace);
2  % Get maximum value of y
3  y_inf = max(y);
4  y_0 = min(y);
5  y_hat =  y_inf-y_0;
6  y_28 = y_0 + 0.28 * y_hat;
7  y_63 = y_0 + 0.63 * y_hat;
8  % Making y unique valued
9  [y_out,i_unique,~] = unique(y,'stable');
10 t_out = t(i_unique);
11 % Calculation of time points where y reaches 28/63 percent
12 t_28 = interp1(y_out,t_out,y_28);
13 t_63 = interp1(y_out,t_out,y_63);
```

Listing 4: This code snippet shows the general way of calculating t_{28} and t_{63}.

First, the y-values for 28% and 63% are computed in matlab. Second, the corresponding t-values are calculated using the built-in `interp1()`-function for interpolation. In line 9 the `unique()`-function returns all values of `y` without repetition. This step is important for the later interpolation.

3.1.3. Results

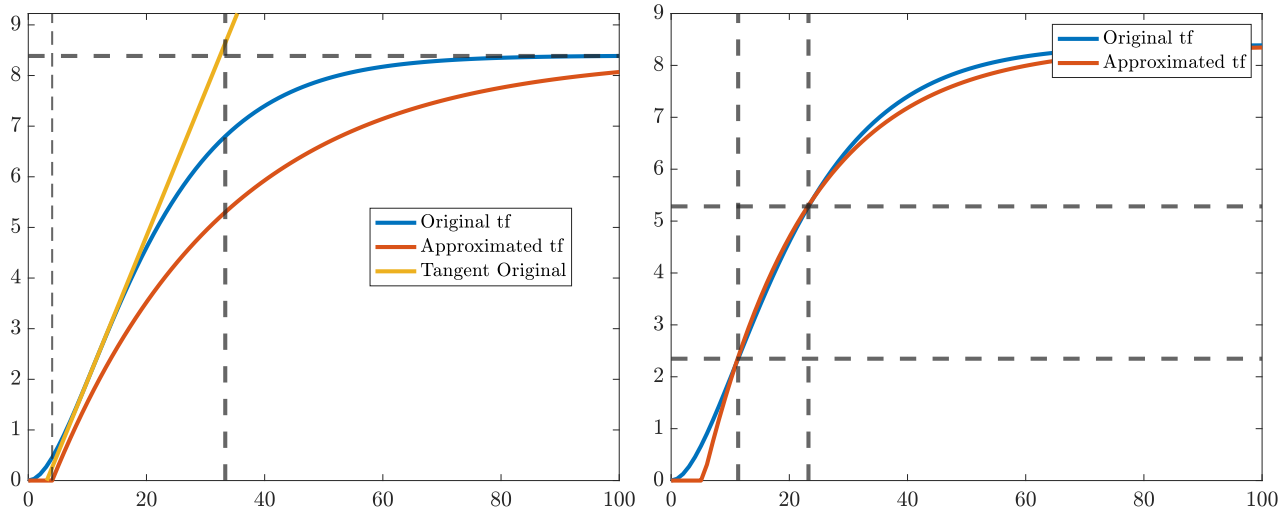
Before presenting the results an Matlab specific issue in PID controller implementation must be adressed. The Simulink PID-block uses the another notation for the PID controller's paraemter, therefore a conversion is needed. This conversion is shown in Listing 5.

```
1 function PID_ideal = matlab_PID_paremters(PID)
2     % Conversion for Simulink PID-block
3     PID_ideal.P = PID.K_p;
4     PID_ideal.I =1/PID.T_I;
5     PID_ideal.D = PID.T_D;
6 end
```

Listing 5: Conversion of the parameters calculated by the previously mentioned algorithms and the values required for Simulink PID block in Matlab.

Table 4: Results of PID controller design using Ziegler-Nichols.

Approximation	v_{wind} [m s^{-1}]	P	I	D	Input-Output pair
max-slope	7	-1.0567	6.0606	1.5152	1-1
two-point	7	-0.4121	10.1010	2.5253	1-1
max-slope	8	-1.0781	8.0808	2.0202	1-1
two-point	8	-0.4554	12.1212	3.0303	1-1
max-slope	9	-0.1923	8.0808	2.0202	1-1
two-point	9	-0.0844	12.1212	3.0303	1-1



(a) Max slope (b) Two-point

Figure 7: Approximation of the transfer function at a wind speed of $8\,\mathrm{m\,s^{-1}}$ and using the input-output pair 1-1

In Figure 7 we can clearly see the different approximation. This results in different control parameters in Table 4. Still, the sign of all control parameters and the differences are small. When the wind speed is changed (using a different transfer function) the proportional factor P of the PID controller changes. The integral I and derivative D terms seem be constant.

3.2. T-sum method

For the input-output pair 2-2 we need to use the T-sum method. The transfer function is of first oder and so Ziegler-Nichols can not be applied. T-Sum tries to find a line where two integrals have the same area. One is the area enclosed by the y-axis and the solution trajectory. The other area is border by the solution trajectory and the y_{max}-value. By shifting the dashed line seen in Figure 8 on the y-axis the optimal point of equal areas is found. The calculated value T_Σ is used to determine the controller parameters.

$$P = \frac{1}{K} \tag{14}$$

$$I = 0.667 \cdot T_\Sigma \tag{15}$$

$$D = 0.167 \cdot T_\Sigma \tag{16}$$

Table 5: Results of PID controller design using T-Sum method.

v_{wind} $[\mathrm{m\,s^{-1}}]$	P	I	D	Input-Output pair
7	0.1724	0.6737	0.1687	2-2
8	0.1479	0.6737	0.1687	2-2
9	0.1240	0.6737	0.1687	2-2

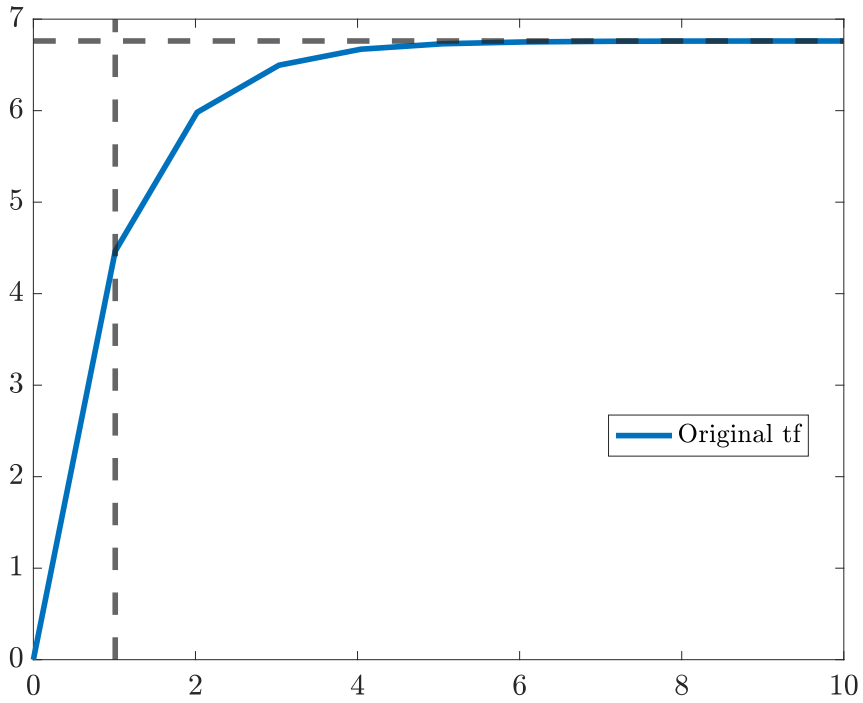


Figure 8: Visualization of the T-sum method for controller design. A controller for the input-output pair 2-2 is designed by using the transfer function for a windspeed of $8\,\mathrm{m\,s^{-1}}$.

Figure 8 shows one possible downside of the T-sum method. When simulating the system the resulting solution is a discrete vector. The T-sum method

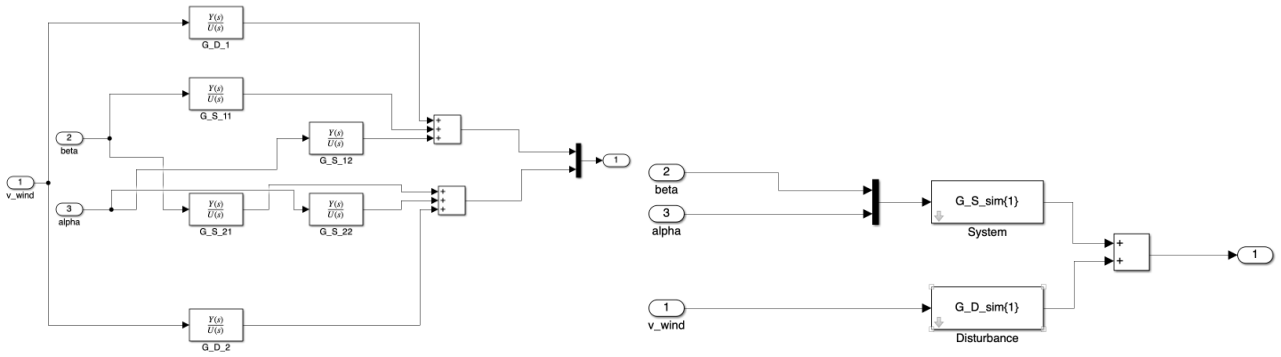
3.3. Implementation in Simulink

In the previous subsection the design of the controller is discussed. Now, the implementation of the closed and open loop is addressed. In contrast to the controller design the closed and open loop is implemented in Simulink.

Open loop

In general two different approaches for the implementation are possible. The transfer function can be arranged as a structured flow diagram (Figure 9a). In this case each transfer function is implemented individually in a single Simulink `tf-block`. One advantage is that the overall structure of the system is visible and all *flows of informations* are visible. Otherwise, this implementation is more complex and therefore more error-prone.

The second case is a more condensed implementation (Figure 9b). In this, only two Simulink `LTI`-blocks are used for the system G and the disturbance G_D.



(a) Implementation as a structure (b) Implementation as blocks

Figure 9: Comparing the two different approaches as Simulink block diagrams. The *structured* implementation (`transfer_function_structured.slx`) reveals much more details about the internal dependencies. A more condensed view is seen for the *block* implementation (`transfer_function_as_block.slx`).

To check the implementation it is good to use both implementation in parallel and check if both are getting the same results. For further investigation the *structured* implementation can be used. This has the advantage that *internal flows* or states can be monitored with Simulinks `scope`- or `out`-blocks.

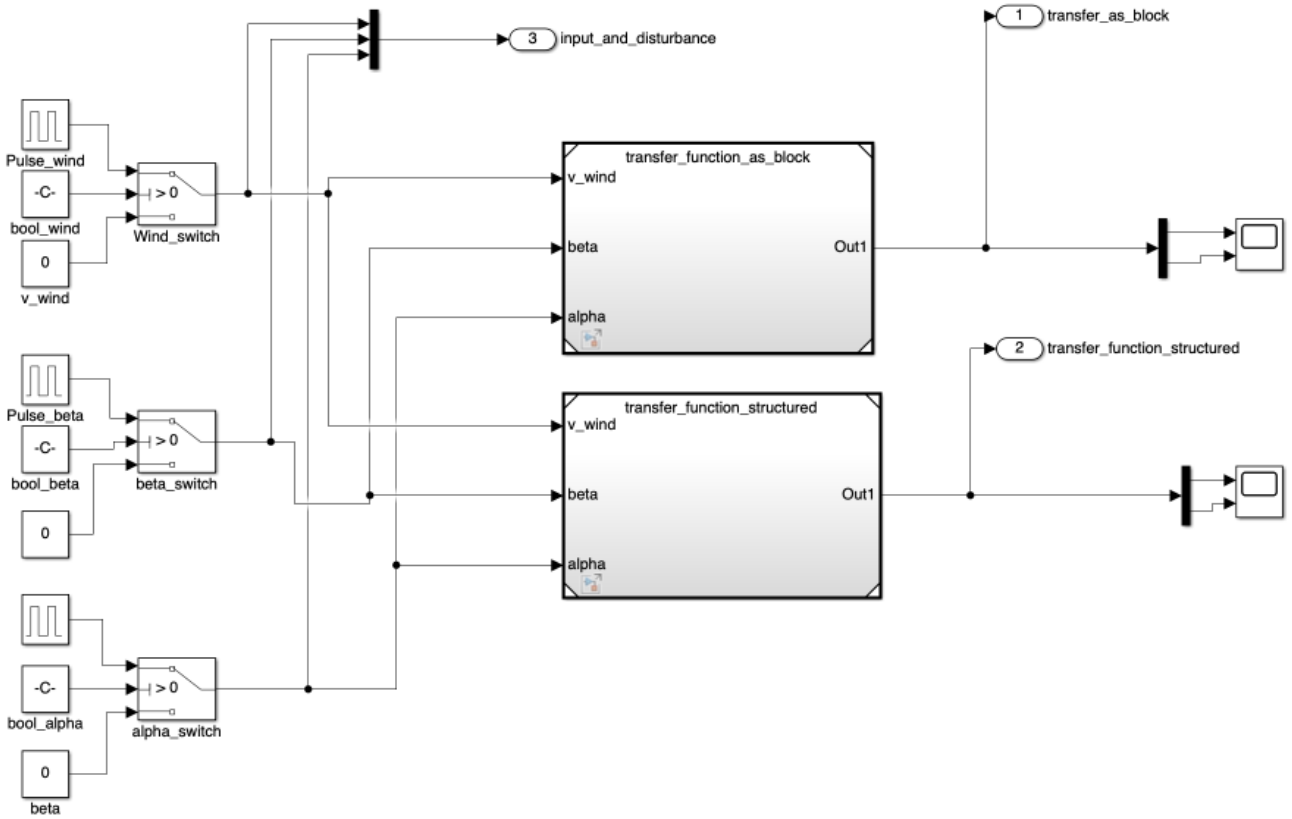


Figure 10: The open loop system with a parallel execution of *structured* and *block* implementation. This figure is showing the `step_response_open_loop.slx`-file.

The Simulink file shown in Figure 10 has three outputs. First outputs return the open loop behaviour for the `block` implementation, the second output for the `structured` implementation. The third `output`-block is to check the inputs into the system.

On the left side of the simulink diagram for each of the three inputs v_{wind} (disturbance), β and α

(from top to down) an `switch`-block is used. These `switch`-blocks allow to choose via a flag set in a Matlab script which input should have a pulse applied to. The pulse is produced by an `pulse`-block in Simulink.

Listing 6 shows how to choose different scenarios depending on flags and `if`-blocks. Lines 7 ff. determine which pulse is given to system. If the variable is set to 0 the input does not have a pulse signal applied to, but it is constant 0. In the given example only β will receive a pulse of magnitude 7° and a width of 600 s after 600 s of simulation time.

```
1  % Setting amplitudes for pulse
2  simP.A_beta = 7;
3  simP.A_alpha = -.1;
4  simP.A_wind = -2;
5
6  % Determine if pulse of zero for inputs and disturbance
7  simP.bool_beta = 1;
8  simP.bool_alpha = 0;
9  simP.bool_wind = 0;
10
11 % Simulation paramters
12 simP.t_simulation = 2000; % Time for simulation [s]
13 simP.T_period = 1200;     % Peroid of pulse [s]
14 simP.T_delay = 600;       % Delay for first pulse [s]
15 simP.pulse_width = 50;    % Pulse width in % of T_Period [%]
```

Listing 6: Setting flags for different combination of input pulses.

Closed loop

When the designed controllers are tested the loop must be closed.

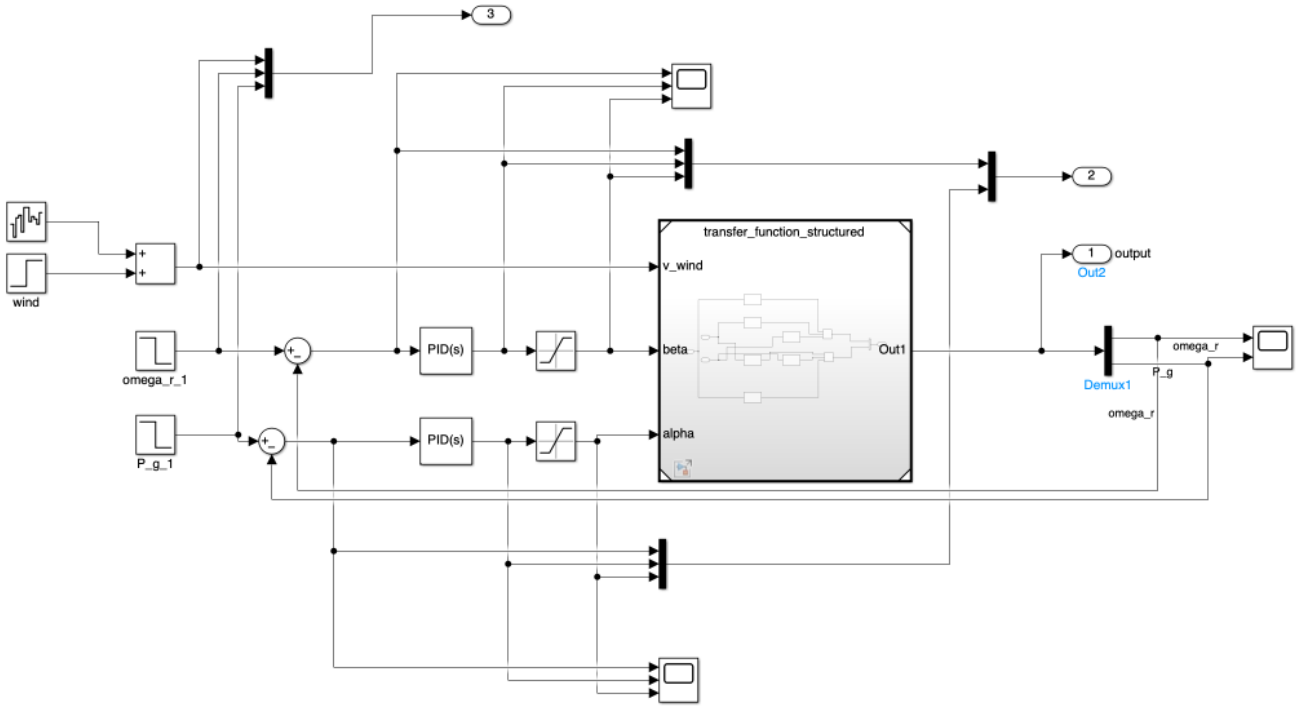


Figure 11: The closed loop with a `PID`-controller block, both reference signals and a possible disturbance due to wind. This figure is showing the `closed_loop_PID.slx`-file.

For possible disturbance due to a change in wind speed a `step`- and a `noise`-block is added. With this setup, not only can the wind change drastically (step change) but also naturally occurring fluctuation can be simulated. After the `PID`-block a `saturation`-block is added. section 4 discusses the need of having a saturation block, which enforces the limitations of input signals shown in Table 2.

4. Results and discussion of different methods

This subsection discusses the performance of the designed controller. Table 6 summarizes all conducted experiments to test the controller. A complete visual summary of all results can is found in the Appendix (Appendix B).

Experiment Number 8 can be described as follows: For the input-output pair 1-1 (PID_{11}) a controller designed with two-point approximation and Ziegler-Nichols and a PID-controller for output pair 2-2 designed by the T-sum method are tested. Both controller are designed for a wind speed of $8\,\mathrm{m\,s^{-1}}$ and tested on the system identified at a wind speed of $8\,\mathrm{m\,s^{-1}}$. The wind is used as a disturbance and has a maximum noise power ξ of 0.5 and a step of magnitude 1 at $t = 1000\,\mathrm{s}$.

Table 6: All conducted experiments with the most important values for controller design and disturbances.

No.	Wind			PID_{11}		PID_{22}		System
	t_{step} [s]	Δv_{wind} [$\mathrm{m\,s^{-1}}$]	ξ [$\mathrm{m\,s^{-1}}$]	Method	v_{wind} [$\mathrm{m\,s^{-1}}$]	Method	v_{wind} [$\mathrm{m\,s^{-1}}$]	
1	0	0	0	ZN-maxS	8	T-sum	8	8
2	0	0	0	ZN-2p	8	T-sum	8	8
3	0	0	0	ZN-maxS	8	T-sum	8	7
4	0	0	0	ZN-2p	8	T-sum	8	7
5	0	0	0.05	ZN-maxS	8	T-sum	8	8
6	0	0	0.05	ZN-2p	8	T-sum	8	8
7	1200	1	0.05	ZN-maxS	8	T-sum	8	8
8	1200	1	0.05	ZN-2p	8	T-sum	8	8
9	1200	0.1	0.005	ZN-maxS	8	T-sum	8	8
10	1200	0.1	0.005	ZN-2p	8	T-sum	8	8

4.1. Recreating benchmark process

Fragoso et al. ([5, p. 14, Figure 13]) is used as a benchmark. To reproduce this figure two step changes are applied. First, at $t = 100\,\mathrm{s}$ a step change of magnitude $A_{P_g} = -1\,\mathrm{W}$ is applied to the generated power P_g. Second, at $t = 800\,\mathrm{s}$ a step change of magnitude $A_{\omega_r} = -200\,\mathrm{rmp}$ is applied to the rotational speed of the blade ω_r.

Figure 13 shows that both Ziegler-Nichols methods design a controller capable of controlling the system. For the max-slope method a oscillating behaviour around the reference value can be seen (Figure 13a) for the rotational speed ω_r after the set-point change. The set-point change in generated power P_g shows less disturbance but also some oscillation around the new set point.

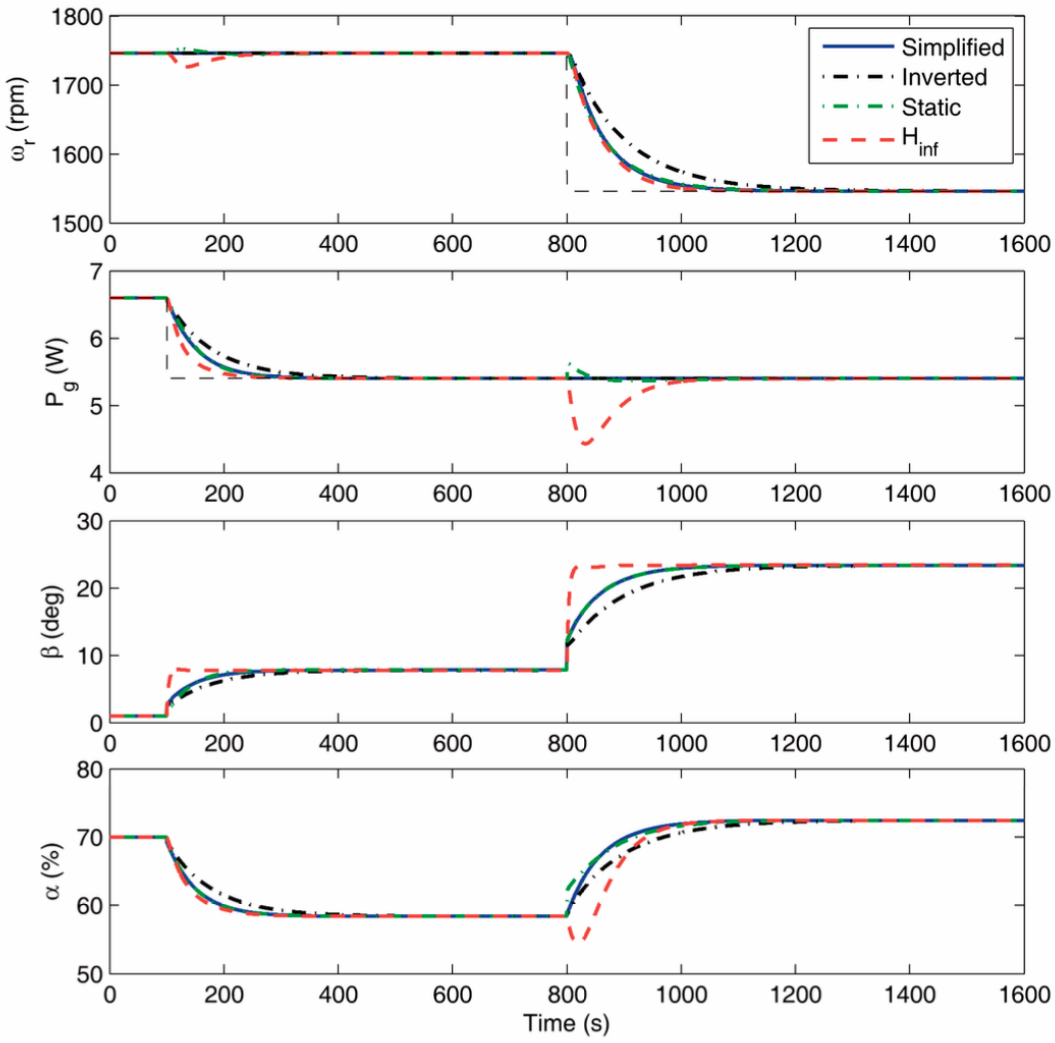
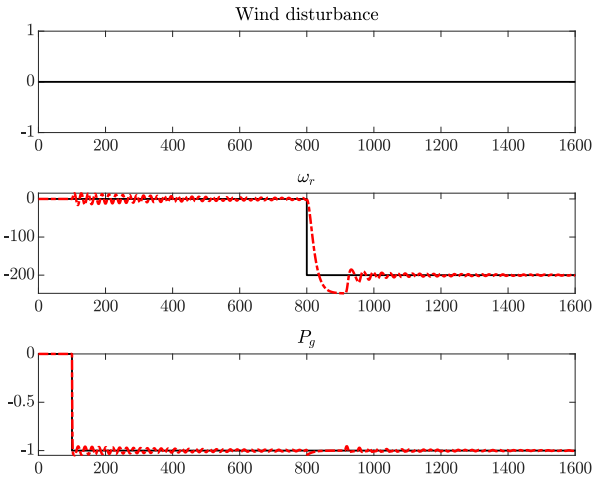


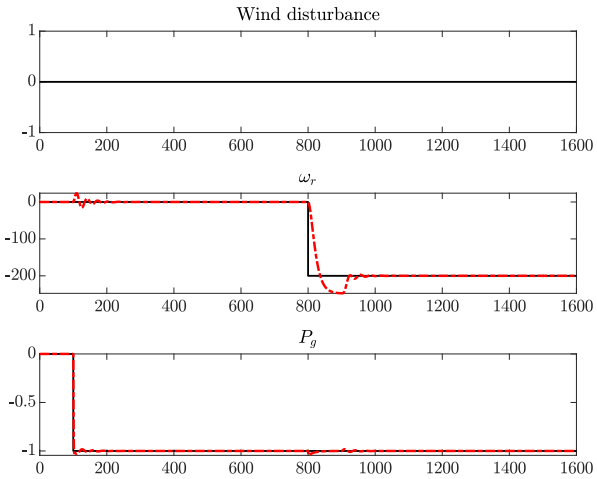
Figure 13. Outputs and control signals of the simulated closed-loop with the nominal model in the transition mode at wind speed 8 m/s.

Figure 12: Figure by [5] used as a benchmark process for designed controllers.

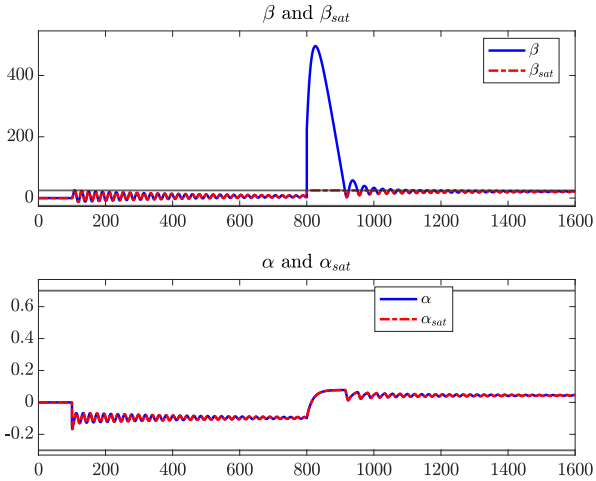
For the duty cycle α the controller output is not saturated. In contrast the pitch angle β is heavily saturated. This shows that the controller is mainly trying to steer the system via the pitch angle, when the rotational speed is changed. The behaviour is consistent with the predicted behaviour by the relative gain.



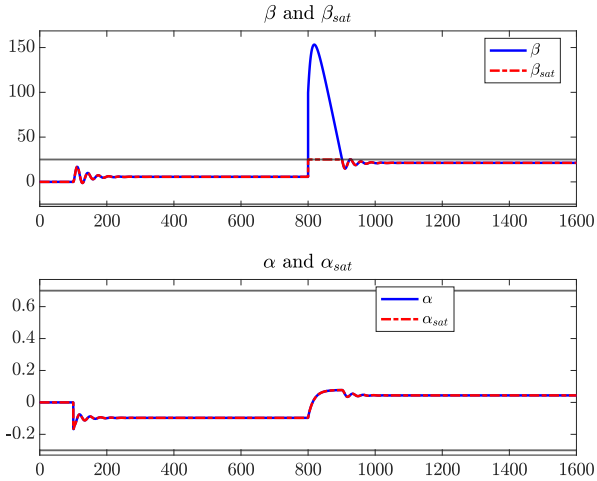
(a) Exp. 1: Wind disturbance, rotational speed of blade and power consumption over time



(b) Exp. 2: Wind disturbance, rotational speed of blade and power consumption over time.



(c) Exp. 1: Blade pitch angle and duty cycle over time before and after saturation.



(d) Exp. 2: Blade pitch angle and duty cycle over time before and after saturation.

Figure 13: Visualization of reference signal and the controller output for experiment 1 and experiment 2.

We now compare the behaviour seen in [5] and in the presented results (Table 7). For the pitch angle the results can be duplicated. When the set-point change is applied to ω_r the duty cycle shows α shows a different behaviour in terms of magnitude, but has the same sign. Thus, global behaviour is also confirmed. One reason for the deviation may also lie in the RGA. Despite the different magnitude, the system can reach the same set-point with a small error. It must be emphasised that all values are read from figures and deviations can occur in this process.

Table 7: Comparing behaviour from [5] and experiments 1 and 2.

Set-point change	Results in Fragoso et al. β	Fragoso et al. α	This work β	This work α
P_g	$+10°$	-10	$+10°$	-10
ω_r	$+20°$	$+15$	$+20°$	$+5$

4.2. PID wind-up

section 3.3 already mentioned the `saturation`-block in the closed loop. We know focus on the PID wind-up. When wind-up occurs the controller is outputting a manipulated variable in an *unrealistic* or physically impossible order of magnitude. This signal is saturated. Therefore the systems input signal is smaller than the controller output. If the reference error is nonetheless not vanishing *fast* enough the controller output is winding up. Which can not be put through to the system because of the saturation.

We look at experiment 7. In experiment 7 at $t = 1200\,\mathrm{s}$ the wind speed increases instantly from $0\,\mathrm{m\,s^{-1}}$ to $0.5\,\mathrm{m\,s^{-1}}$. The controller tries to concur this disturbance by increasing the pitch angle β. Due to the saturation the controller can not *act hard enough*. The error increases and the controller increases the output (higher pitch angle). The pitch angle exceed 360° multiple times, which is a physical not plausible behaviour. There are several anti-wind-up strategies to prevent an increasing error.

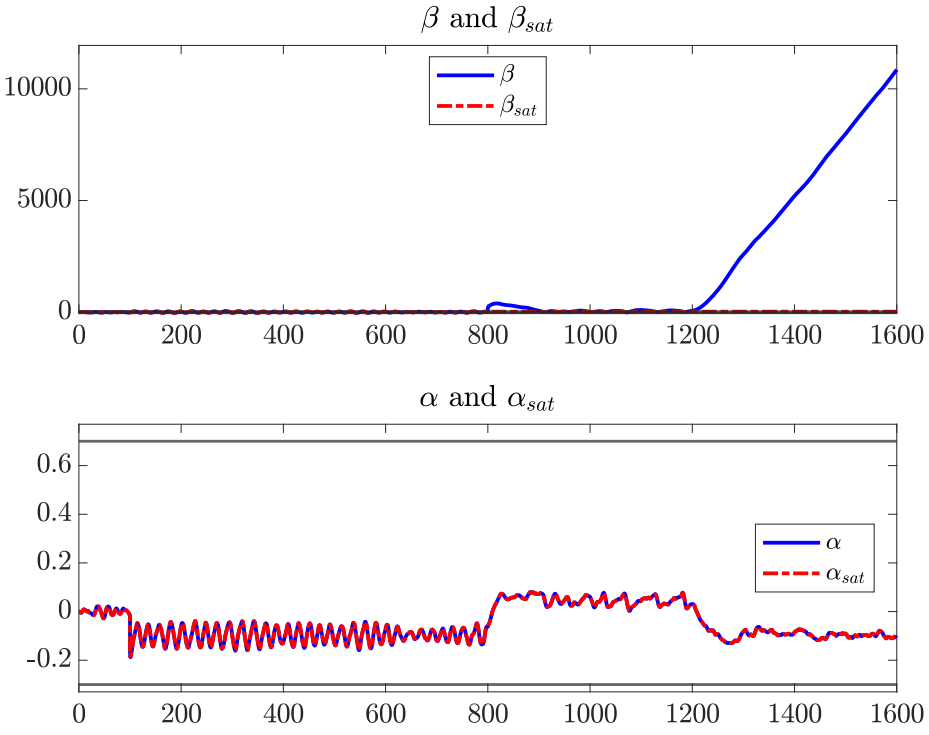
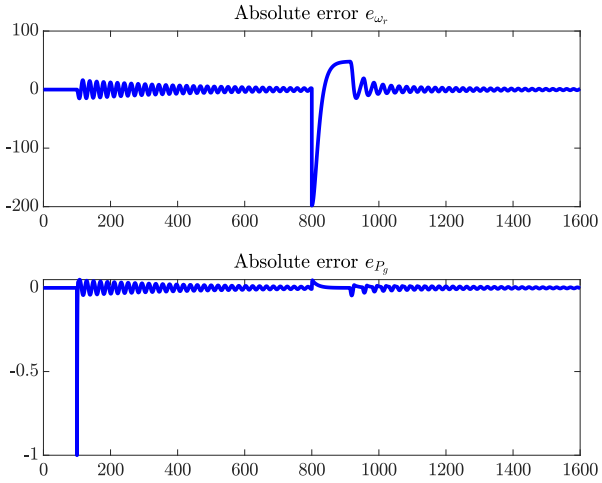


Figure 14: Wind-up of a PID controller seen in experiment 7.

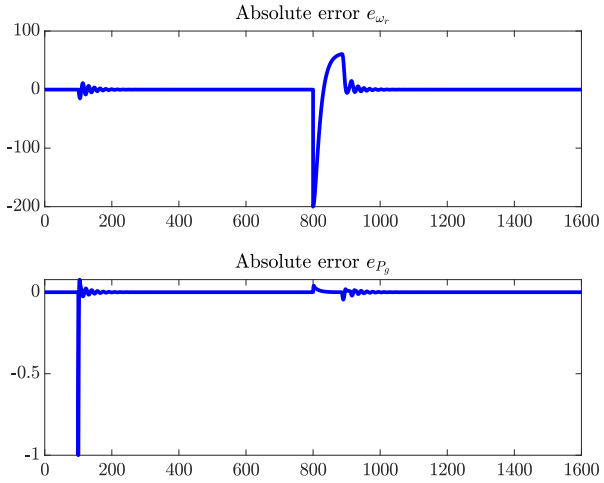
4.3. Using controller outside of identified transfer function

This subsections answers the question if a controller designed for a specific wind speed can be used in other *wind speed domains*. To answer the question a controller designed using transfer function at a wind speed of $8\,\mathrm{m\,s^{-1}}$ is used with the identified system at a wind speed of $6\,\mathrm{m\,s^{-1}}$. Looking at the results of

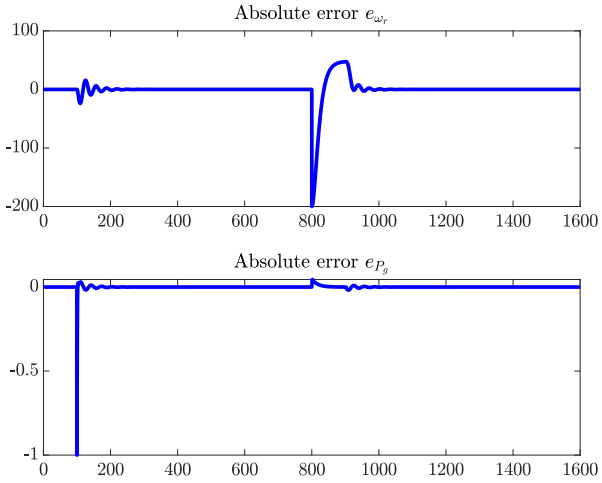
experiments 3 and 4 and comparing it to the results of experiments 1 and 2, lead to the conclusion that a controller can be used *outside* of its wind speed domain (Figure 15). Experiment 1 and experiment 3 use the same parameters, in experiment 3 only the controller is designed for a system with a wind speed of $7\,\mathrm{m\,s^{-1}}$ instead of $8\,\mathrm{m\,s^{-1}}$. The same holds for the experiment pair 2 and 4.



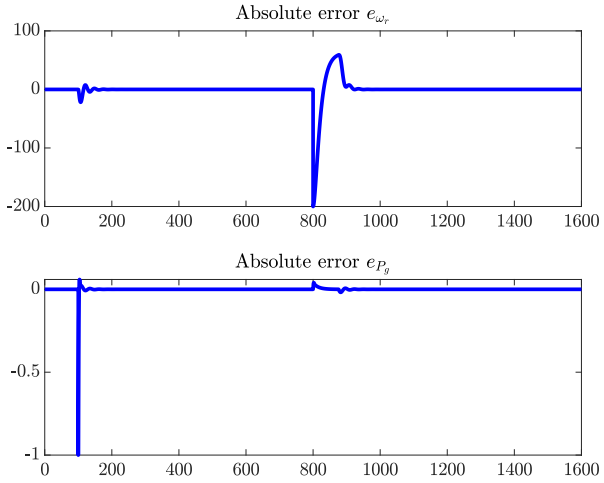
(a) Exp. 1: Absolute error over time



(b) Exp. 3: Absolute error over time



(c) Exp. 2: Absolute error over time



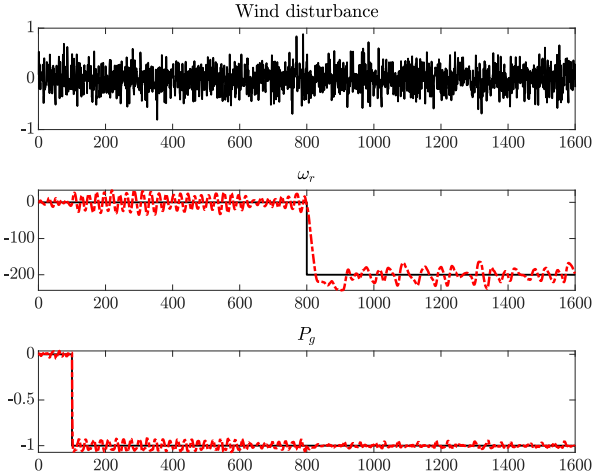
(d) Exp. 4: Absolute error over time

Figure 15: Comparison of the time course of the absolute error for experiments 1 to 4.

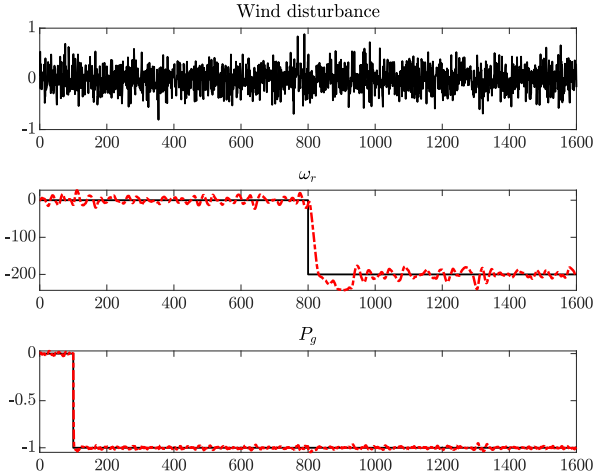
4.4. Introduction of noise

For the first four experiments no noise was introduced. For a real life application this a unrealistic scenario especially for a naturally occurring phenomena like wind. To address this a white noise and lastly a step change (boe) is introduced. This section focuses on the controllers capability to deal with noise.

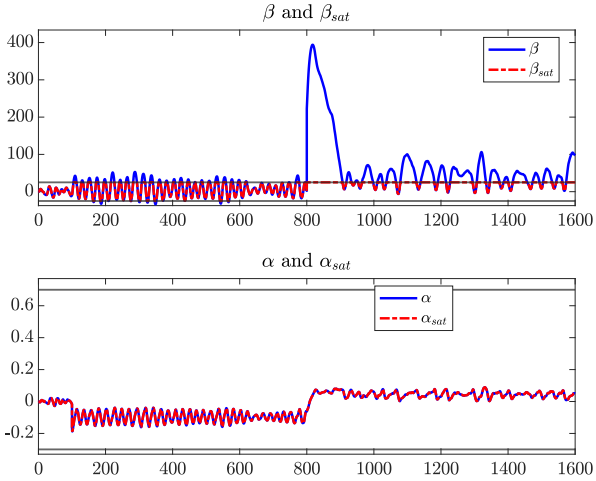
First, we will look at the introduction of a white noise with a *noise power* of $\xi = 0.05\,\mathrm{m\,s^{-1}}$ (experiments 5 and 6) and analyze the controller output and the set-point derivation.



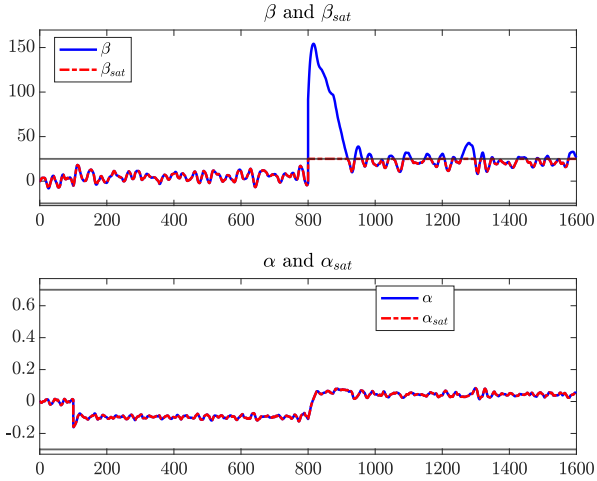
(a) Exp. 5: Disturbance and control variable



(b) Exp. 6: Disturbance and control variable



(c) Exp. 5: Controlled variable



(d) Exp. 6: Controlled variable

Figure 16: Influence of white noise to controller performance

Both controllers are showing good results. Due to the high frequency of the white noise the controller reacts with many and also high frequent control inputs. In a real life application this could cause problems. A controller can not be arbitrarily fast with its outputs due to limitation in computation and especially of the actuators. A blade can not change its pitch angle in millis, in particular for real blades, this would cause a great strain. A filter could be used to filter the high frequent wind changes.

A. Matlab Code

A.1. Ziegler-Nichols-Methods

A.1.1. Computing infliction point

```
function [slope, intcpt] = calc_infliction_point_slope(y,t)
    %% Find the inflection point and tangent
    % Source: https://de.mathworks.com/matlabcentral/answers/295156-how
       -to-find-the-inflection-point-of-a-curve-in-matlab
    % gradient of step response
    dydt = gradient(y);
    % Find y and t of highest gradient
    % Get only the unique values
    [dydt,i_unique,~] = unique(dydt,'stable');
    t_out = t(i_unique);
    y_out = y(i_unique);
    t_infl = interp1(dydt, t_out, max(dydt));
    y_infl = interp1(t_out, y_out, t_infl);
    % Calculate slope
    slope  = interp1(t_out, dydt, t_infl);
    % Find y-interception
    intcpt = y_infl - slope*t_infl;

end
```

Listing 7: Matlab code for calculating the infliction point.

A.1.2. Computing control parameters

```
function [K, T, T_d,t_interception] = calc_gain_time_const_time_delay(y
   ,t,tngt)

    % %Find the T_d and T
    % Calculate the intercept of slope with K
    K = max(y);
    % Get t-value when tangents hits K
    % First we calculate the index in t. With this index the time t
    % can be calculated
    t_index_interception = find(tngt>=max(y),1,'first');
    t_interception = t(t_index_interception);
    t_index_T_d = find(tngt>=0,1,'first');
    % Calculation of T_d and T
    T_d = t(t_index_T_d);
    T = t_interception-T_d;


end
```

Listing 8: Computation of T_d, T and K in Matlab.

A.1.3. Two-point method

```
[y,t] = step(G,t_linspace);
% Get maximum value of y
```

```
y_inf = max(y);
y_0 = min(y);
y_hat =  y_inf-y_0;
y_28 = y_0 + 0.28 * y_hat;
y_63 = y_0 + 0.63 * y_hat;
% Making y unique valued
[y_out,i_unique,~] = unique(y,'stable');
t_out = t(i_unique);
% Calculation of time points where y reaches 28/63 percent
t_28 = interp1(y_out,t_out,y_28);
t_63 = interp1(y_out,t_out,y_63);

%% Calculation of T, T_d
T = 3/2 * (t_63-t_28);
T_d = t_63 - T;
K = y_inf;
```

Listing 9: Implementation of two-point method in Matlab.

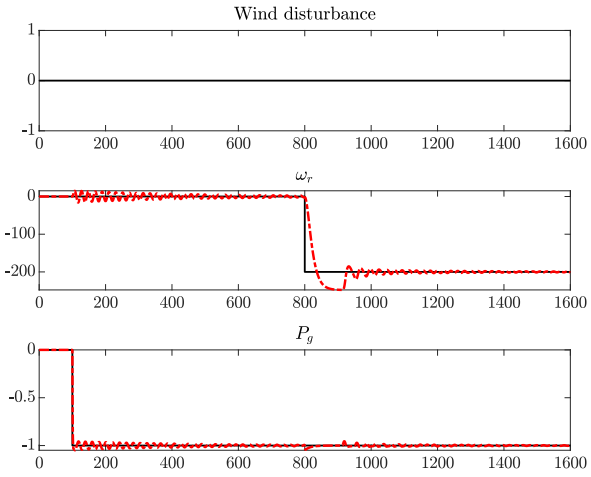
T-sum Method

```

[y,t_out] = step(G,t_linspace);
% Get K and plot it
K = max(y);
% Find T_sum
N = size(t_out,1);
delta_A = zeros(1,N);
delta_T = gradient(t_out);


for n=1:N
    T_sum = t_out(n);
    T_delta = t_out(end)-T_sum;
    A_1 = sum(y(1:n).*delta_T(1:n));
    A_2_hat = sum(y(n+1:end).*delta_T(n+1:end));
    A_2 = K*T_delta - A_2_hat;
    delta_A(n) = abs(A_1-A_2);
end
[~, index_min_delta_A] = min(delta_A);
T_sum = t_out(index_min_delta_A);

% Calculate PID values
PID.K_p = 1/K;
PID.T_I = 0.667 * T_sum;
PID.T_D = 0.167 * T_sum;
```

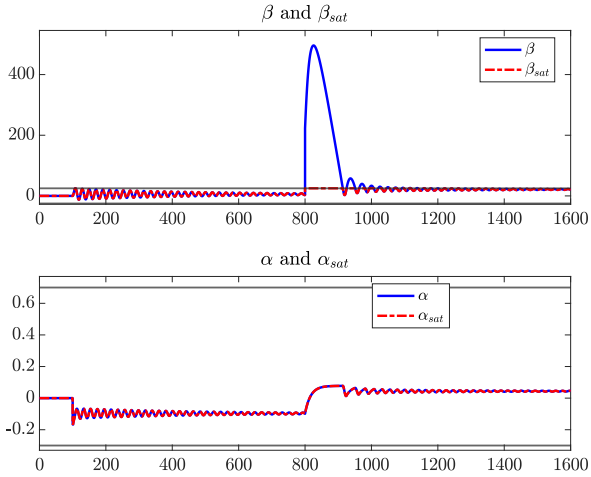
Listing 10: Implementation of two-point method in Matlab.

B. Closed loop results

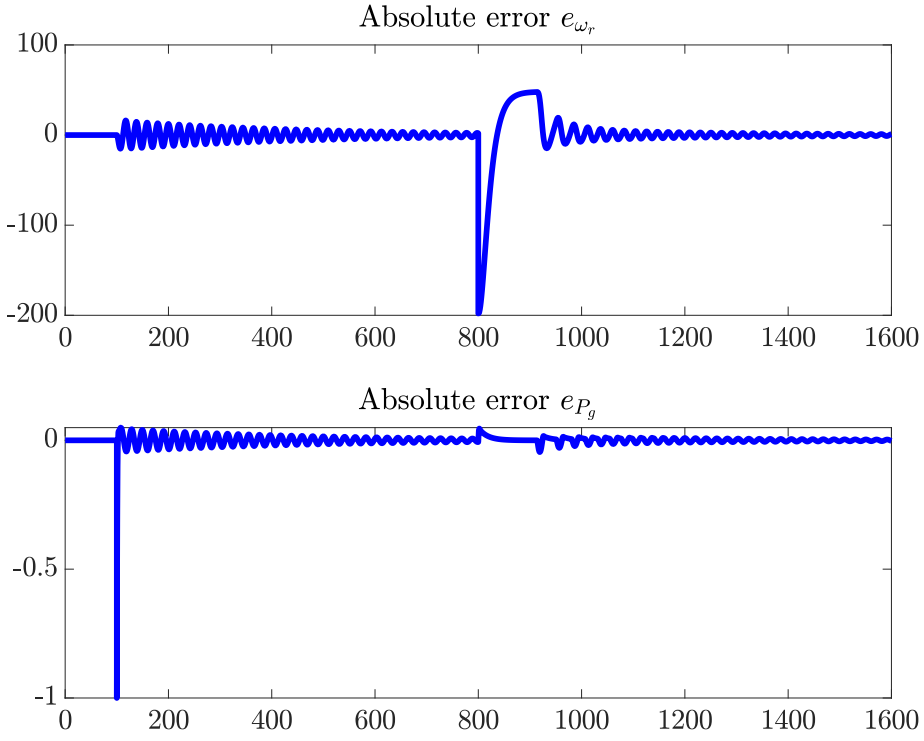
B.1. Experiment 1



(a) Wind disturbance, rotational speed of blade and power consumption over time



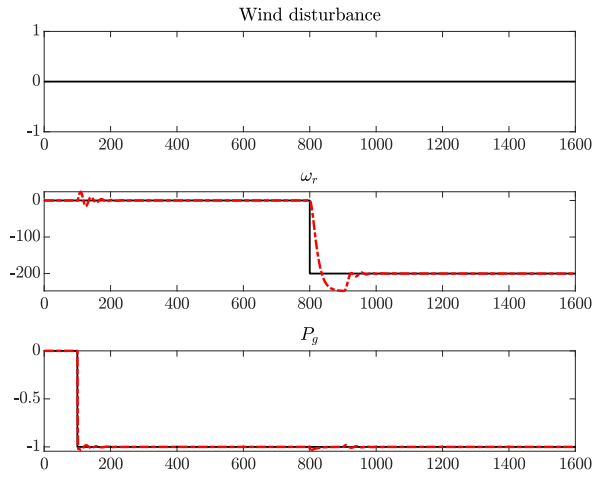
(b) Blade pitch angle and duty cycle over time before and after saturation.



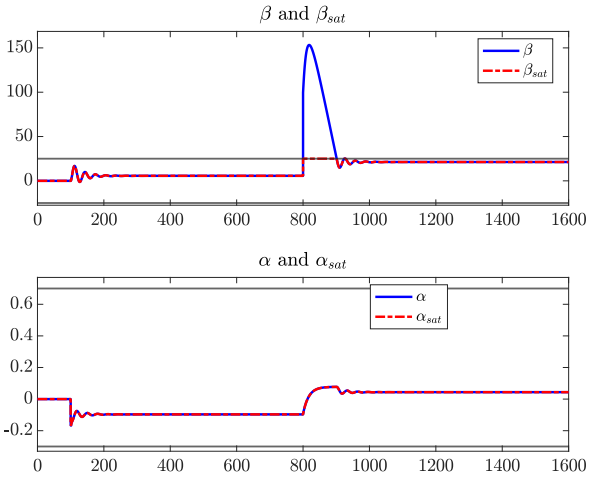
(c) Relative error over time.

Figure 17: Visualization of reference signal, the controller output and the absolute error

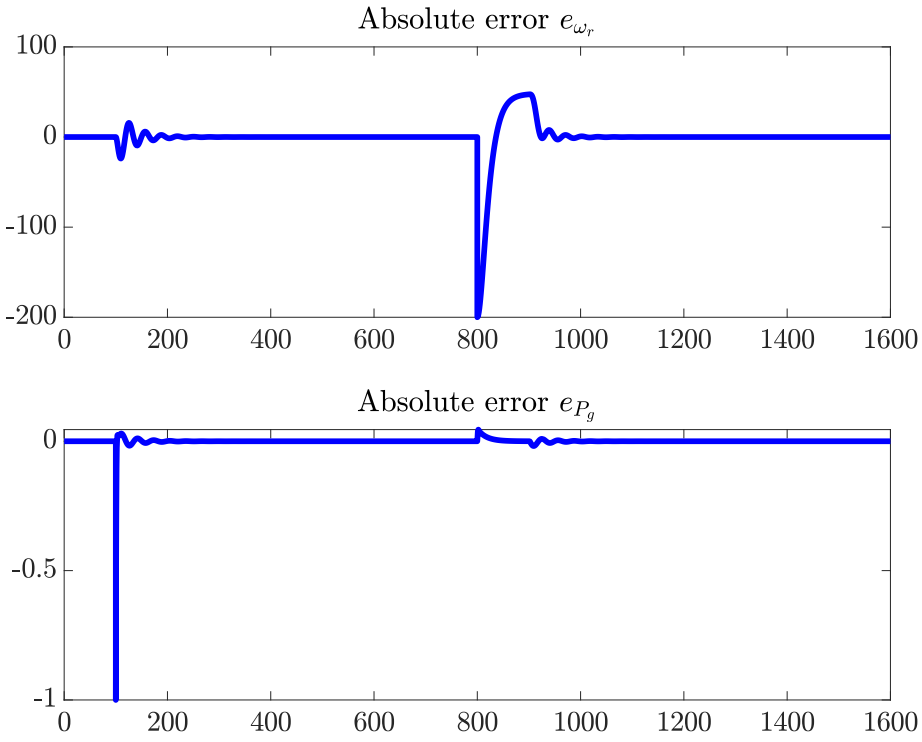
B.2. Experiment 2



(a) Wind disturbance, rotational speed of blade and power consumption over time



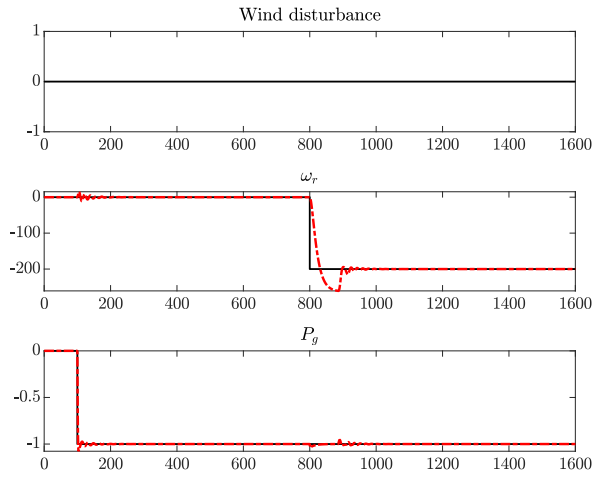
(b) Blade pitch angle and duty cycle over time before and after saturation.



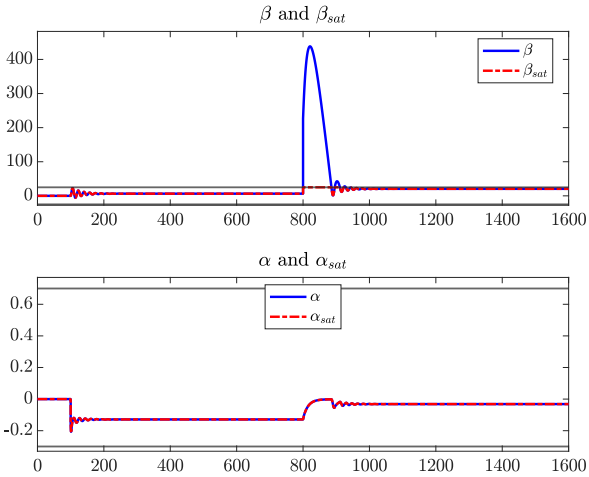
(c) Relative error over time.

Figure 18: Visualization of reference signal, the controller output and the absolute error

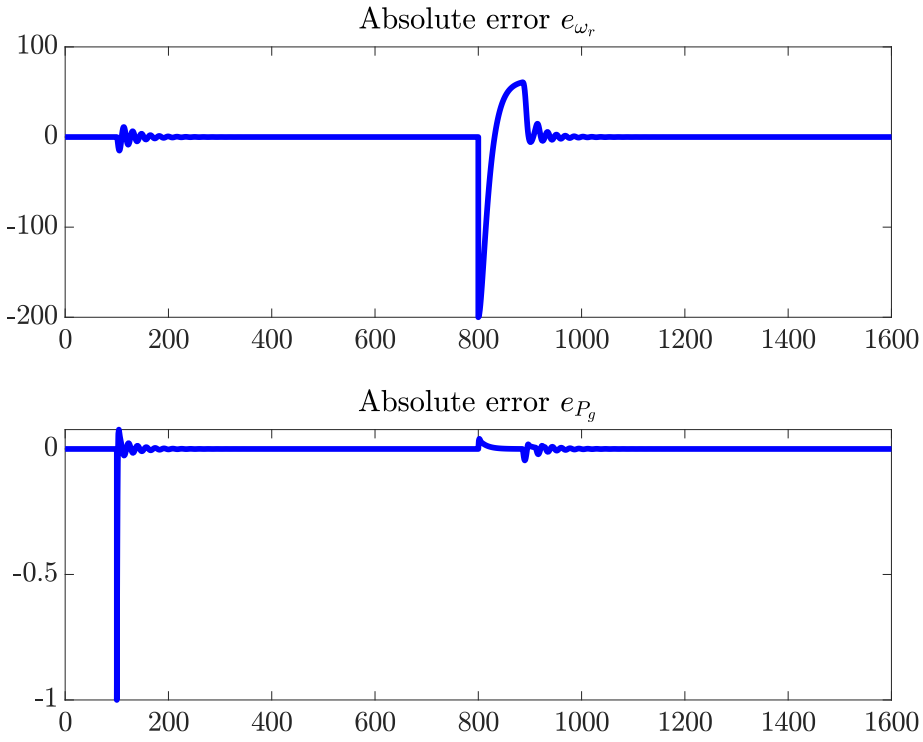
B.3. Experiment 3



(a) Wind disturbance, rotational speed of blade and power consumption over time



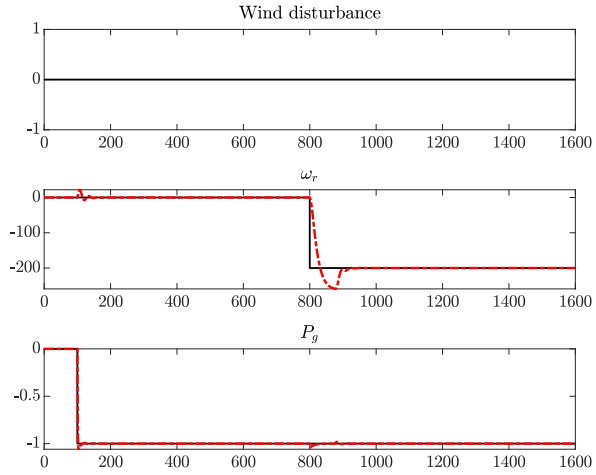
(b) Blade pitch angle and duty cycle over time before and after saturation.



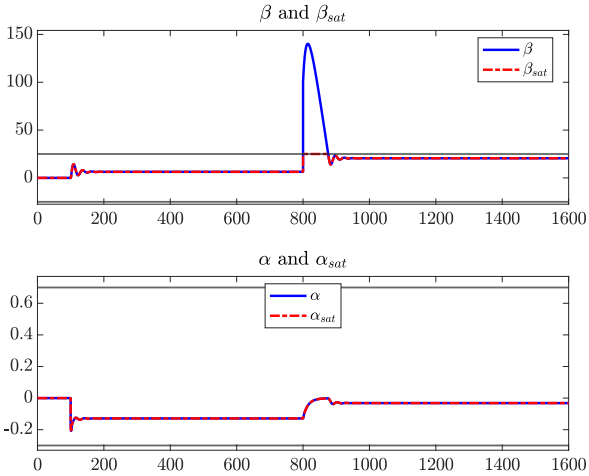
(c) Relative error over time.

Figure 19: Visualization of reference signal, the controller output and the absolute error

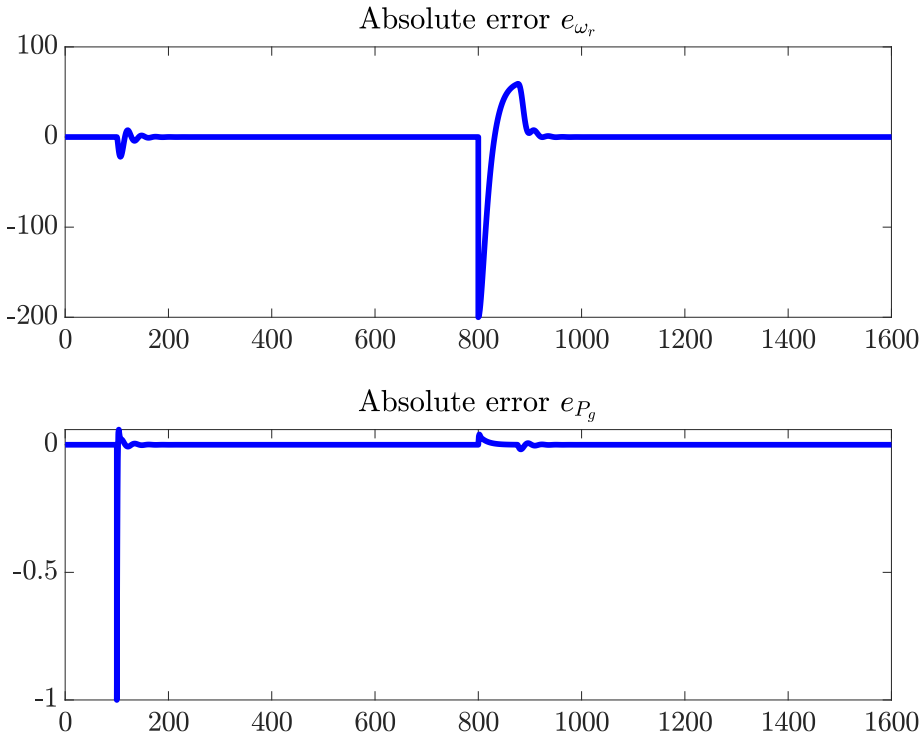
B.4. Experiment 4



(a) Wind disturbance, rotational speed of blade and power consumption over time



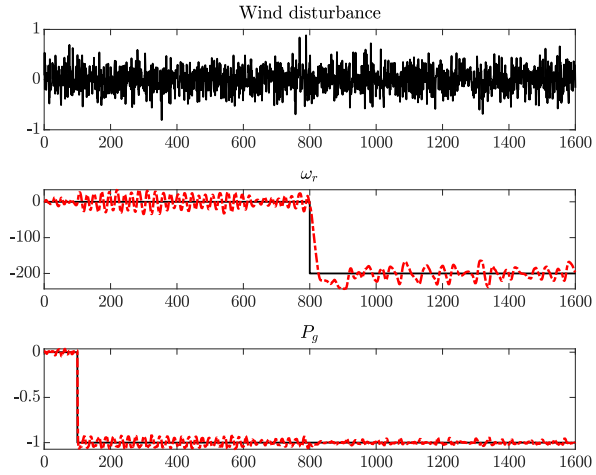
(b) Blade pitch angle and duty cycle over time before and after saturation.



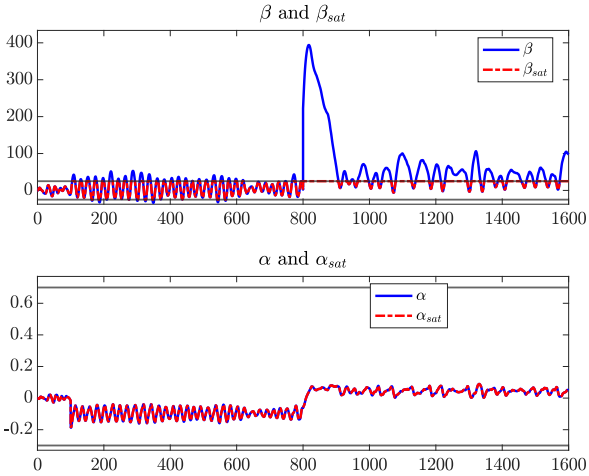
(c) Relative error over time.

Figure 20: Visualization of reference signal, the controller output and the absolute error

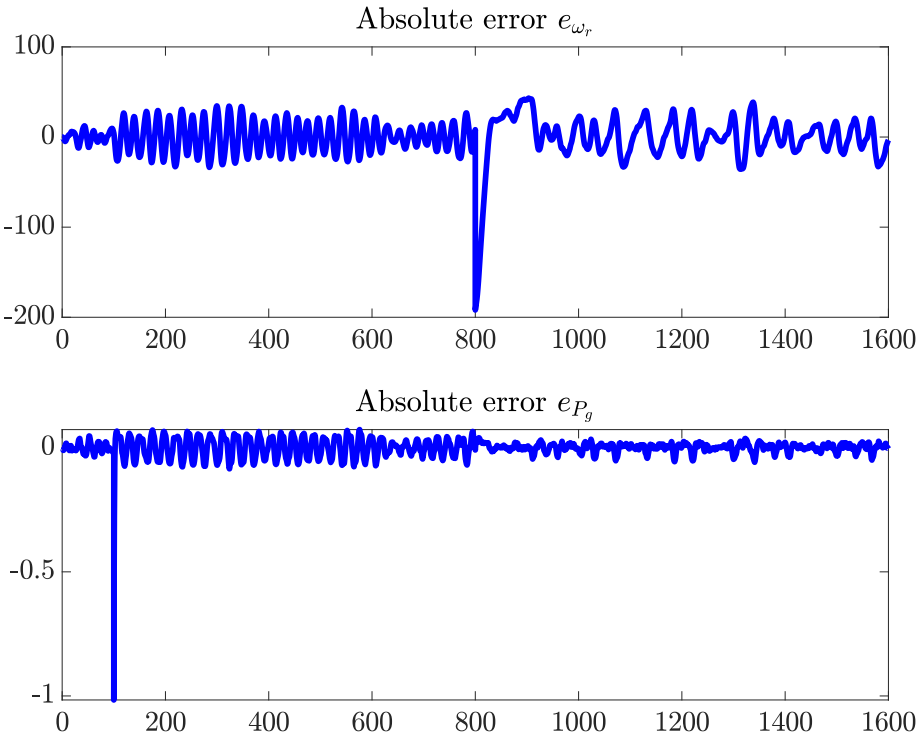
B.5. Experiment 5



(a) Wind disturbance, rotational speed of blade and power consumption over time



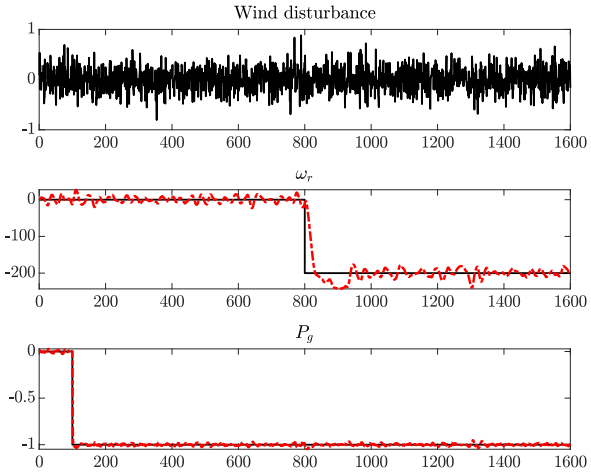
(b) Blade pitch angle and duty cycle over time before and after saturation.



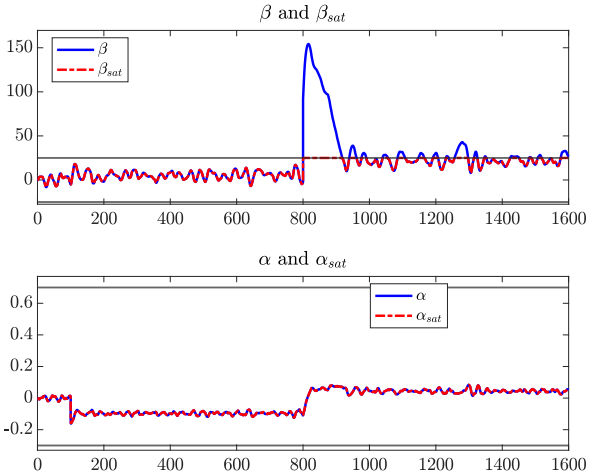
(c) Relative error over time.

Figure 21: Visualization of reference signal, the controller output and the absolute error

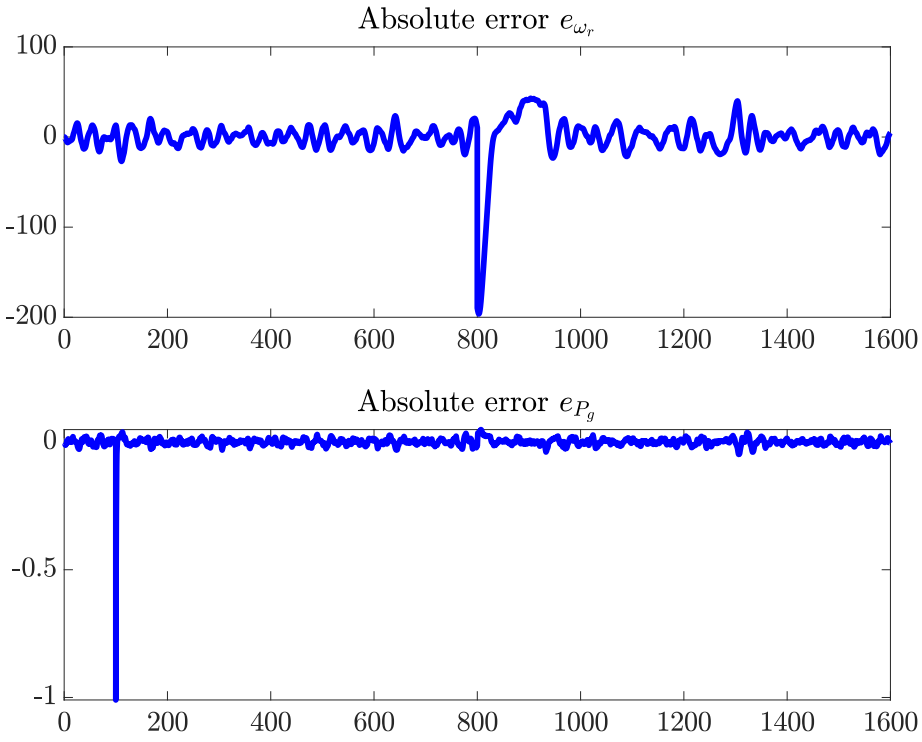
B.6. Experiment 6



(a) Wind disturbance, rotational speed of blade and power consumption over time



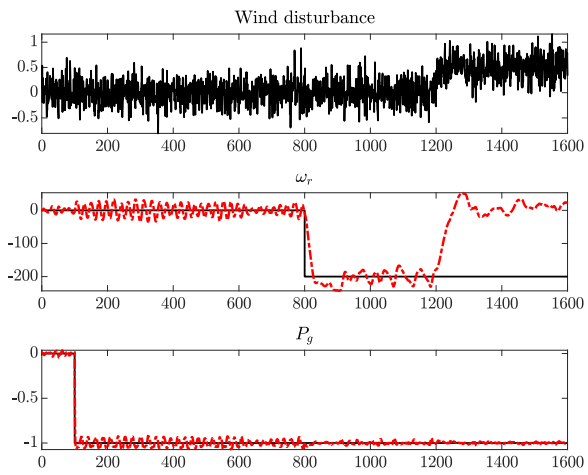
(b) Blade pitch angle and duty cycle over time before and after saturation.



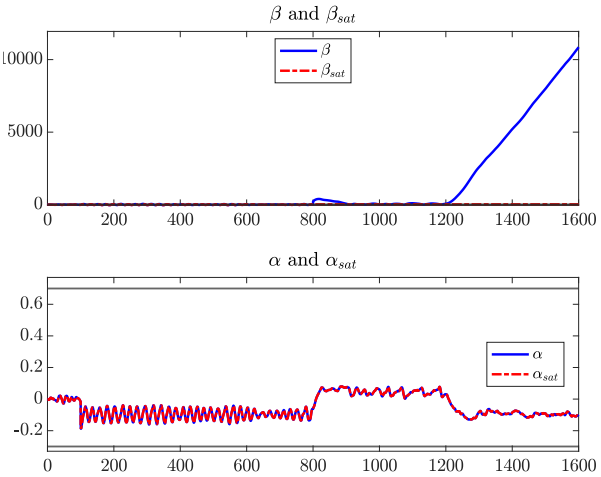
(c) Relative error over time.

Figure 22: Visualization of reference signal, the controller output and the absolute error

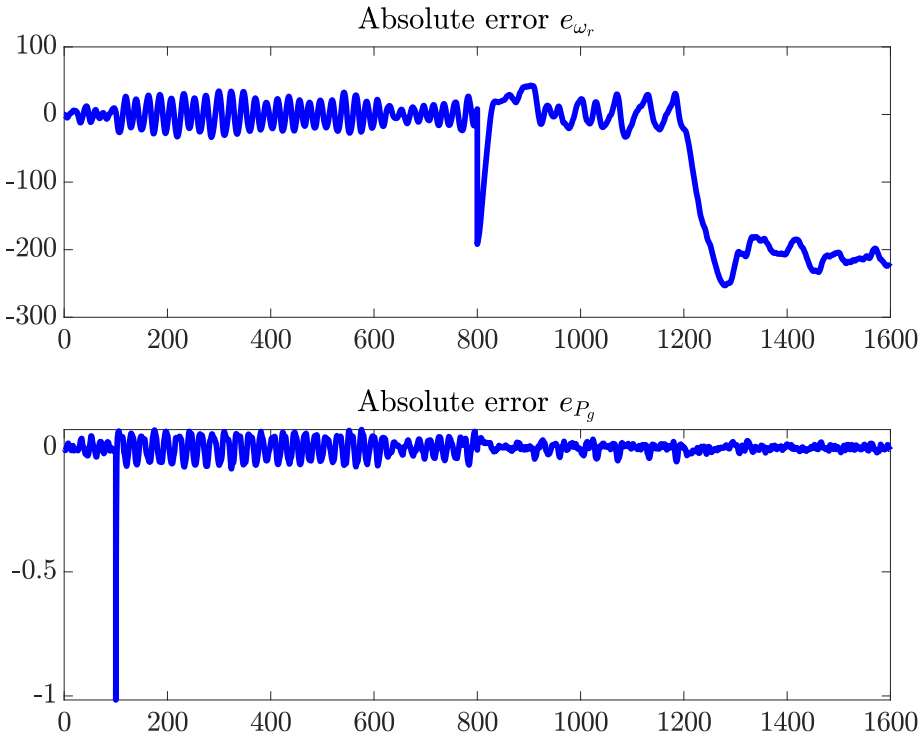
B.7. Experiment 7



(a) Wind disturbance, rotational speed of blade and power consumption over time



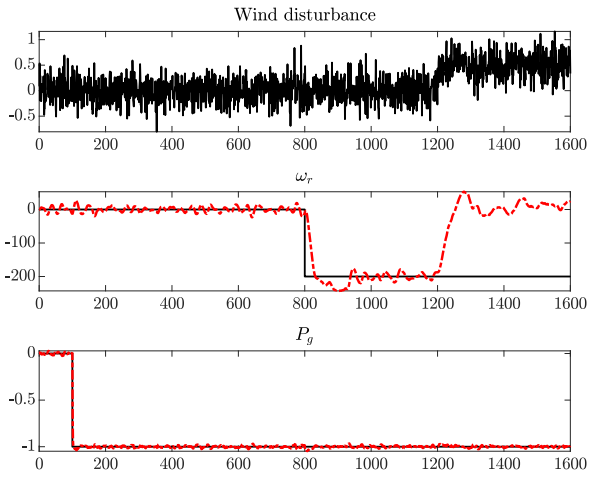
(b) Blade pitch angle and duty cycle over time before and after saturation.



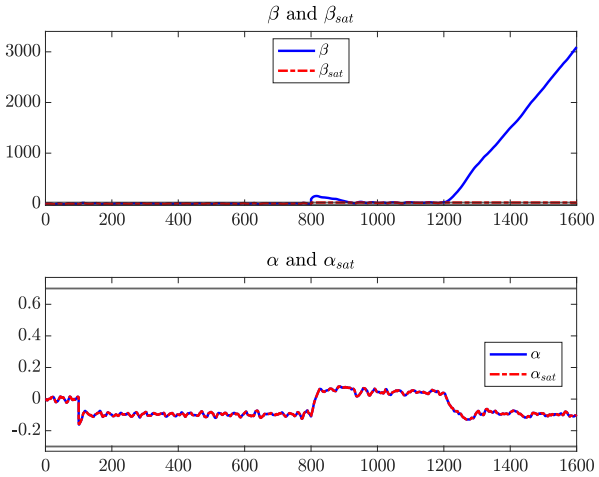
(c) Relative error over time.

Figure 23: Visualization of reference signal, the controller output and the absolute error

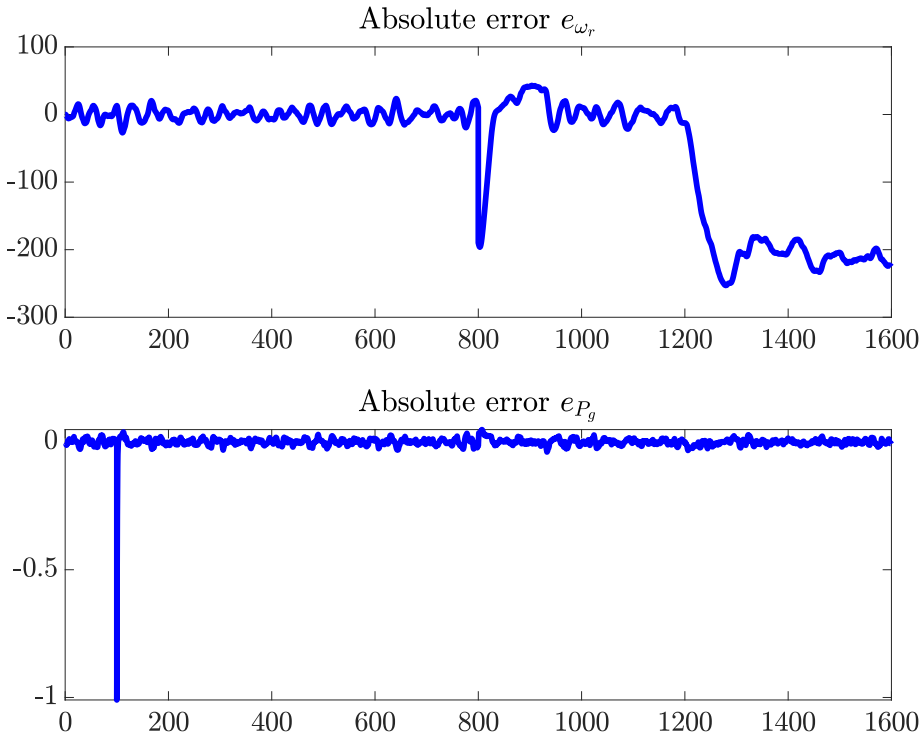
B.8. Experiment 8



(a) Wind disturbance, rotational speed of blade and power consumption over time



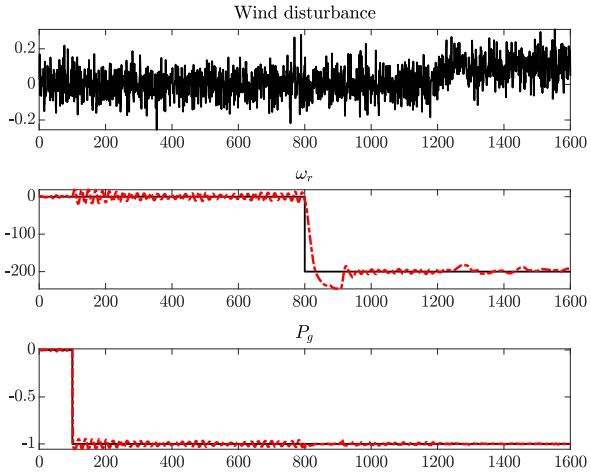
(b) Blade pitch angle and duty cycle over time before and after saturation.



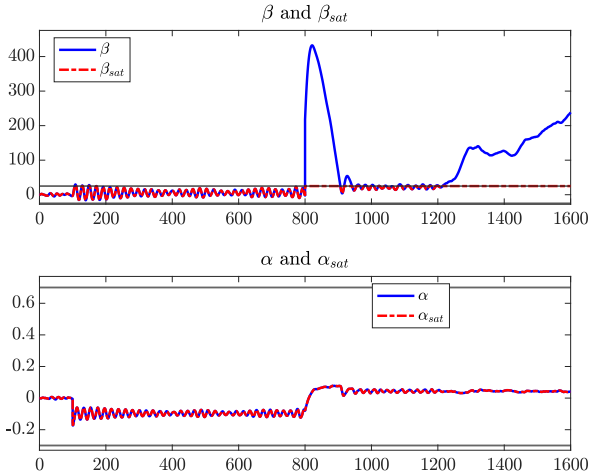
(c) Relative error over time.

Figure 24: Visualization of reference signal, the controller output and the absolute error

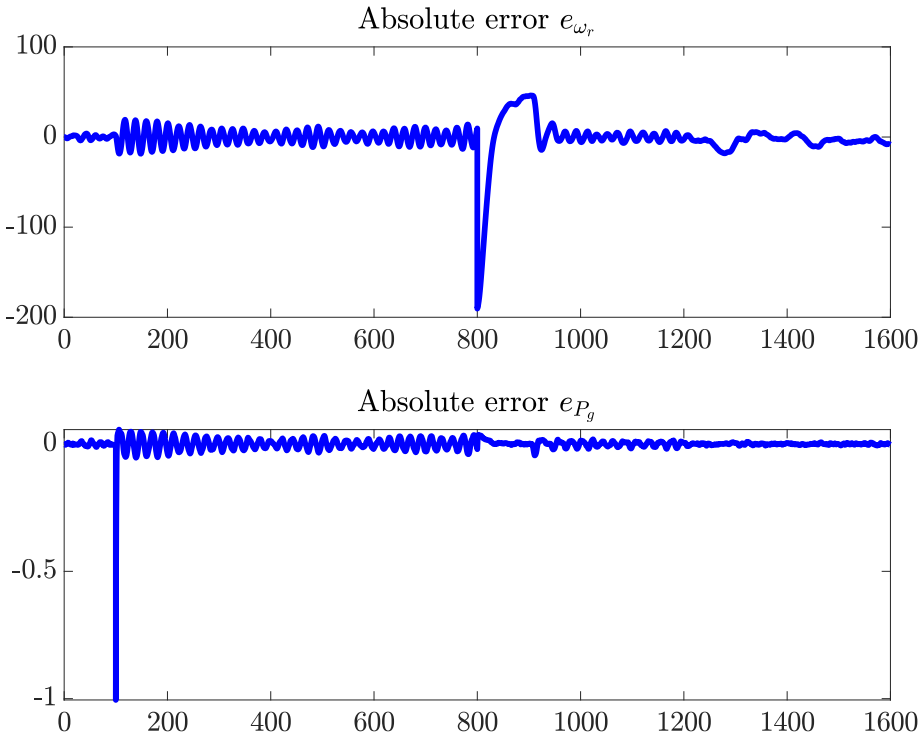
B.9. Experiment 9



(a) Wind disturbance, rotational speed of blade and power consumption over time



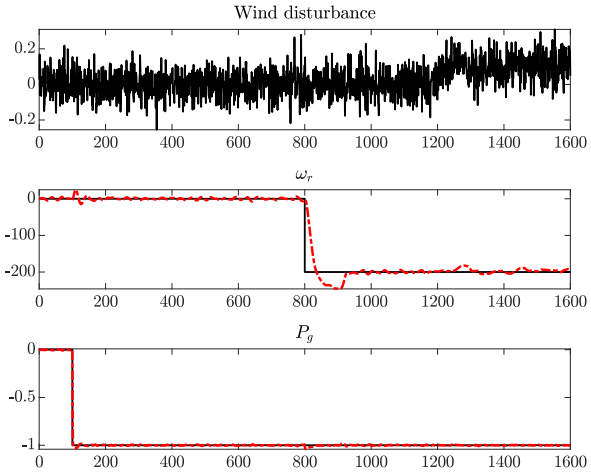
(b) Blade pitch angle and duty cycle over time before and after saturation.



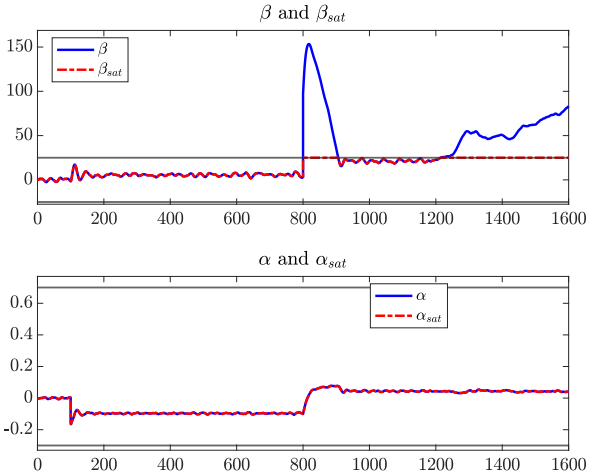
(c) Relative error over time.

Figure 25: Visualization of reference signal, the controller output and the absolute error

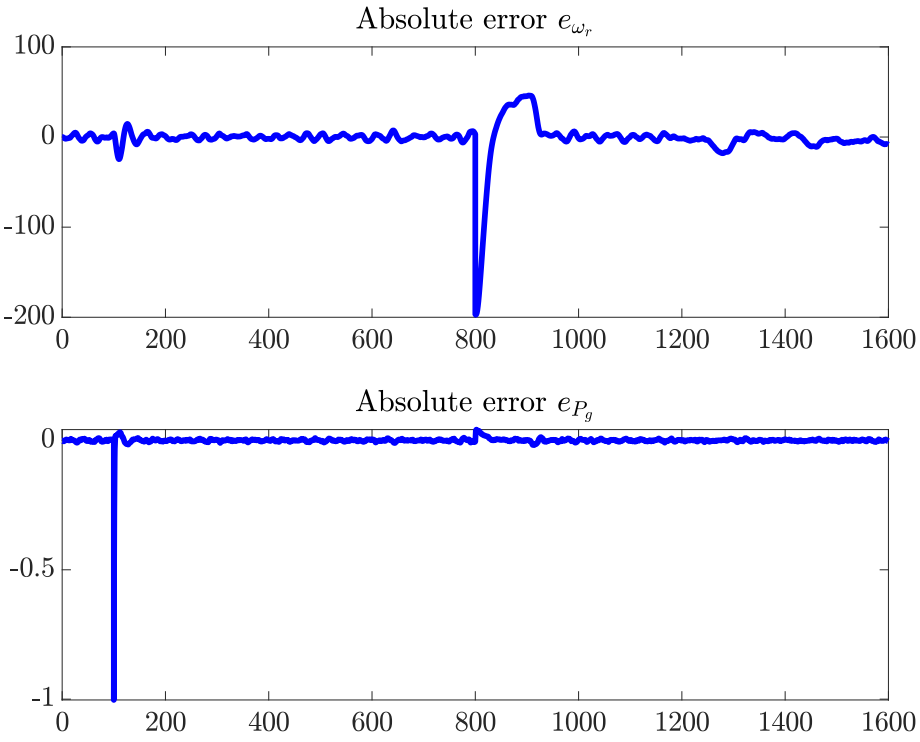
B.10. Experiment 10



(a) Wind disturbance, rotational speed of blade and power consumption over time



(b) Blade pitch angle and duty cycle over time before and after saturation.



(c) Relative error over time.

Figure 26: Visualization of reference signal, the controller output and the absolute error

References

[1] José Miguel Adánez, Basil Mohammed Al-Hadithi, and Agustín Jiménez. Wind Turbine Multivariable Optimal Control Based on Incremental State Model. 20(6):2075–2087, nov 2018.

[2] International Energy Agency. Renewable electricity generation by source (non-combustible), World 1990-2020. www.iea.org, 2022.

[3] Ilknur Dissli-Kienle. Lecture Process Control OvGU, 2023.

[4] Sergio Fragoso. *Diseño de sistemas de control multivariable mediante redes de desacoplo: aplicación al control de aerogeneradores*. Phd thesis, Universidad de Córdoba, UCOPress, 2015. Available at `https://helvia.uco.es/xmlui/handle/10396/13734`.

[5] Sergio Fragoso, Juan Garrido, Francisco Vázquez, and Fernando Morilla. Comparative Analysis of Decoupling Control Methodologies and H_∞ Multivariable Robust Control for Variable-Speed, Variable-Pitch Wind Turbines: Application to a Lab-Scale Wind Turbine. *Sustainability*, 9(5), 2017.

[6] Silvio Simani. Overview of modelling and advanced control strategies for wind turbine systems. *Energies*, 8(12):13395–13418, 2015.

[7] Yuan Yuan, Xu Chen, and J. Tang. Multivariable robust blade pitch control design to reject periodic loads on wind turbines. *Renewable Energy*, 146:329–341, 2020.