Vebjørn Malmin
Halvor Ødegård Teigen

Reinforcement Learning and Predictive Safety Filtering for Floating Offshore Wind Turbine Control

A Step Towards Safe AI

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
June 2021

Vebjørn Malmin
Halvor Ødegård Teigen

Reinforcement Learning and Predictive Safety Filtering for Floating Offshore Wind Turbine Control

A Step Towards Safe AI

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Preface

This report is written as a part of TTK4900 Master Thesis and concludes our Masters Degree in Cybernetics and Robotics at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). The work was supervised by Professor Adil Rasheed.

We would like to thank Thomas N. Larsen for being a great discussion partner and for his input throughout the work. Finally, we would like to thank Professor Adil Rasheed for always being available for questions and for his guidance and supervision during this project.

This work was done as a part of the Operational Control for Wind Power Plants (Grant No.: 268044/E20) project funded by the Norwegian Research Council and its industrial partners (Equinor, Vestas, Vattenfall) in collaboration with SINTEF Digital.

Trondheim, 7.6.2021
Halvor Ødegård Teigen
Vebjørn Malmin

Abstract

Artificial intelligence is seen as one of the most significant leaps in technology in recent years, with the subcategory of reinforcement learning showing exceptional results for previously thought-to-be impossible problems. However, one of the major concerns with reinforcement learning methods is related to a complete lack of guarantees on their performance and safety. This has limited their use in safety critical and high-stakes real-life applications. To this end, our research attempts to address the issue by developing a framework for combining reinforcement learning with an adaptation of model predictive control called the predictive safety filter. The framework, capable of guaranteeing stability and constraint satisfaction, will bridge the gap between research and real-world applications.

The framework is applied to a floating offshore wind turbine, due to their increasing importance and significance in both Norwegian and international industry. The complicated and constantly evolving dynamics of wind turbines promote the use of learning-based methods, eliminating the need for expensive and time consuming derivations of mathematical models. We show that applying our method can ensure constraint satisfaction both during the training and after the deployment of a reinforcement learning agent controlling the turbine. We also show that the predictive safety filter in some cases accelerates the learning.

The framework **RL-PSF** (Teigen and Malmin, 2021) is written in Python, and is publicly available as open-source code under the GNU General Public License. The implementation is designed to be highly modular, in the sense that it can be used with any learning-based controller and is not domain or application specific. This will enable further research in the field of safe artificial intelligence.

Sammendrag

Kunstig intelligens blir sett på som et av de mest betydningsfulle sprangene innen teknologi de siste årene. Underkategorien forsterkende læring har vist eksepsjonelle resultater for problemer som tidligere var antatt umulige. Samtidig er en av de største bekymringene med metoder innen forsterkende læring knyttet til en fullstendig mangel på garantier for ytelse og sikkerhet. Dette har begrenset bruken av metoden i sikkerhetskritiske applikasjoner. Forskningen som presenteres i denne teskten prøver å løse problemet ved å utvikle et rammeverk som kombinerer forsterkende læring med en tilpasset versjon av modell prediktiv kontroll. Kontrolalgoritmen har fått navnet prediktivt sikkerhetsfilter, og brukes i rammeverket på grunn av filters evne til tilfredstille beskraninger og garantere stabilitet. Rammeverket har som mål å bygge bro mellom forskning og virkelige applikasjoner for sikker bruk av forsterkende læring.

Grunnet den store veksten innen utenskjærs vindkraft, i både norsk og internasjonal industri, ble rammeverket anvendt på en flytende vindturbin for å undersøke dets ytelse og anvendbarhet. Den kompliserte dynamikken til vindturbiner fremmer bruken av læringsbaserte metoder, noe som eliminerer behovet for dyre og tidkrevende utledninger av matematiske modeller. Vi viser at bruk av rammeverket kan sikre mot brudd av beskrankninger når en forsterkningslæringsagent styrer turbinen - både under trening og etter utplassering. I tillegg viser vi at det prediktive sikkerhetsfilteret kan akselererer læringen i noen tilfeller.

Rammeverket **RL-PSF** (Teigen and Malmin, 2021) er skrevet i Python, og er offentlig tilgjengelig som åpen kildekode under GNU General Public License. Implementeringen er designet for å være svært modulær, i den forstand at den kan brukes med en hvilken som helst læringsbasert regulator og ikke er domene- eller applikasjonsspesifikk. Dette vil muliggjøre videre forskning innen fagområdet sikker kunstig intelligens.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

AI	Artificial Intelligence
DVT	Dynamic Vortex Theory
FOWT	Floating Offshore Wind Turbine
IEA	International Energy Agency
LLCP	Log-Log Convex Program
LMI	Linear Matrix Inequality
MDP	Markov Decision Process
ML	Machine Learning
MPC	Model Predictive Control
MPPT	Maximum Power Point Tracking
NLP	Non-Linear Program
ODE	Ordinary Differential Equation
PGML	Physics Guided Machine Learning
PPO	Proximal Policy Optimization
PSF	Predictive Safety Filter
RK	Runge-Kutta
RL	Reinforcement Learning
RWT	Reference Wind Turbine
SDP	Semi-Definite Program
SDS	Sustainable Development Scenario
TD	Temporal Difference
VSVP	Variable Speed Variable Pitch

Other

$\alpha, \beta, \ldots$	Italic: Scalars

$\mathbb{I}_{\leq a}$	The set of all natural number in the range $[0, a]$
$\mathbb{I}_{a,b}$	The set of all natural number in the range $[a, b]$
$\mathbb{R}^{+}$	The set of all real non-negative numbers
$\mathbf{A}, \mathbf{B}, \ldots$	Capital bold font: Matrices
$\mathbf{a}, \mathbf{b}, \ldots$	Bold font: Column vectors
$\mathcal{A}, \mathcal{B}, \ldots$	Calligraphic: Sets
$a, b, \ldots$	Italic: Scalars

Operators

$\mathbf{f}\vert_{x}$	The function $\mathbf{f}$ evaluated at the point x
$\mathrm{col}_i(\mathbf{A})$	The i-th col of the matrix $\mathbf{A}$
$\mathrm{row}_i(\mathbf{A})$	The i-th row of the matrix $\mathbf{A}$

Chapter 1

Introduction

Reinforcement Learning (RL) has gained substantial traction in recent years, superseding the performance of state-of-the-art methods. The field has rendered previously thought-to-be impossible control tasks, such as autonomous vehicles, possible. The advancement of this technology has vast potential and could revolutionize the industry throughout. However, its black-box nature and the lack of safety guarantees raise concerns in industry adaptation and limit its use-case in real-world applications.

Safety certification through adapted control methods is emerging as a way to harness the power of RL while retaining the formal proofs of stability. The field is novel and in need of further exploration. This thesis explores one of these methods applied to an offshore wind turbine system and provides a framework for further exploration of its potential in similar applications.

1.1 Motivation and background

Floating offshore wind turbines

Reports from the Intergovernmental Panel on Climate Change (IPCC, 2021) show that climate change and global warming is in fact real, and may have enormous consequences for the planet. The reports also point out that "*Limiting global warming to* $1.5^{o}C$ *rather than* $2^{o}C$ *above pre-industrial levels would make it markedly easier to achieve many aspects of sustainable development, with greater potential to eradicate poverty and reduce inequalities*". With the energy sector and industrial processes accounting for 76% of global greenhouse gas emissions in 2018 (World Resources Institute, 2021), it is clear that this is a sector in need of renewable alternatives.

The International Energy Agency's (IEA) Sustainable Development Scenario (SDS) aims to realize the goal of $1.5°$C and outlines a significant transformation in the global energy system where wind power is one of the main focuses. According to the IEA, offshore wind power has increased significantly in recent years and will have to continue to accelerate to reach the Sustainable Development Goals. As seen in Figure 1.1.1, the SDS outlines a significant increase in offshore wind power generation within 2030. The IEA reports that expansion is accelerating in China, and the European Union has returned to growth after a slowdown in 2018, with record installations in 2019. The European Energy Research Alliance (EERA) has also started the JP WIND program to provide strategic leadership for research and to support the European wind energy industry. EERA's DeepWind conference presents the state of the art in on-going research and innovation related to deep-sea offshore wind farms and is hosted annually by SINTEF in Trondheim.

Wind power is one of the fastest-growing energy sources globally, with a 53% growth in 2020 (Global Wind Energy Council, 2021). It is not a particularly new technology, and the Norwegian Statkraft has developed on-shore wind turbines for around 20 years. However, these on-shore wind farms have created opposition due to their many environmental issues. Despite its renewable energy generation, other environmental concerns like wildlife disruption, noise pollution, and visual intrusion on nature have raised concerns about their development. Bringing the turbines offshore eliminates many of these concerns and has become a focus area with projects like Vineyard Wind 1 (*Vineyard Wind 1*, 2021), Empire Wind 2, and Beacon Wind 1 (NTB, 2021). Norway is a major contributor to this development, with Equinor landing contracts for the latter two and other projects like the Norwegian Offshore Wind Cluster. The Norwegian Offshore Wind Cluster was established in 2016 to be the most vital supply chain for floating offshore wind worldwide.

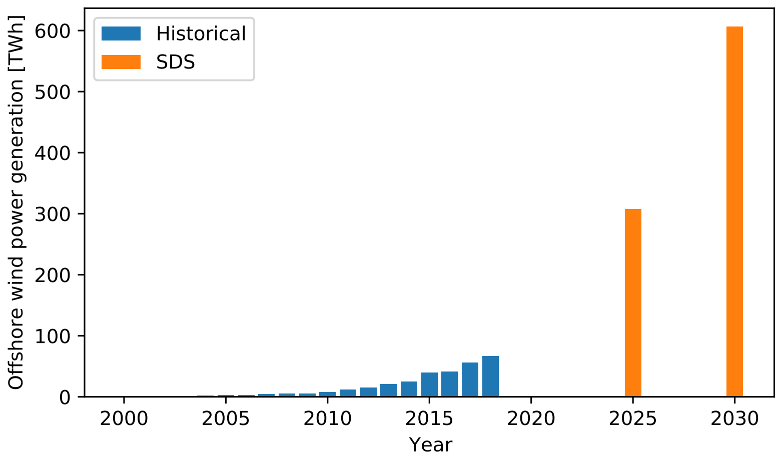


Figure 1.1.1: Offshore wind power generation according to the Sustainable Development Scenario, 2000-2030, (IEA, 2020).

Deep reinforcement learning

Offshore wind turbines include several control systems to stabilize the turbine and keep the power generation optimal. The downside of using traditional control methods is the need for complex mathematical models of the dynamics (which is rarely fully known). Individual controllers for each subsystem, and a switch between controllers for low and high wind speeds, are also common (Jafarnejadsani et al., 2012). Using model-free RL removes the need for complex models and explicit behavioral programming and creates a global controller for the whole system. The RL agent learns the end-to-end connection between observations and actions through the principle of trial and error, which has shown remarkable results in applications such as games (Silver et al., 2016), robotics (Niroui et al., 2019), and natural language processing (He et al., 2016).

Unlike some advanced model-based control strategies, like Model Predictive Control (MPC), RL does not have to use a model or solve an optimization problem at each timestep once the agent is trained and deployed. It exclusively uses the learned policy, which makes it more suitable for real-time applications. Another advantage of RL's learning-based nature is that an agent can continue to learn after deployment if desired. The ultimate end goal is to deploy a trained agent onto a physical Floating Offshore Wind Turbine (FOWT) to use its knowledge of the simulated turbine to control the real-world one safely. The belief is then that the agent can take advantage of its robustness and, optionally, further adapt the policy to the nonlinearities of the real-world turbine.

Although RL as a concept has proven powerful for various problems, there are major concerns regarding safety requirements and constraint satisfaction due to its black-box nature. This causes problems during training of the agent due to the *trial and error* methodology of RL, but also after deployment because of explainability issues. The low interpretability has reduced the applicability of RL in real-world, safety-critical applications and is one of the most significant drawbacks of this approach.

Safe RL and Predictive Safety Filtering

In the process of learning the dynamics of a system, an RL agent has to explore the environment by bringing the system to a variety of states. This exploration often leads the agent to be in unwanted and unsafe states. Even an optimal policy (in the eyes of the agent) may perform poorly in some cases (Taha, 2013). The agent's perception of optimality is highly dependent on how we define it through the so-called reward function. An important thing to note is that optimal long-term performance does not necessarily avoid the rare occurrences of unwanted adverse outcomes. Several methods to increase safety and constraint satisfaction for real-world applications have been presented by researchers to reduce the risk of this. Some of them relate to changing the RL internally, while others apply a more modular approach, e.g., by filtering the signals from the RL externally.

García and Fernández (2015) present some of the different approaches to safe RL that are related to changes internally. They present two main categories for the solution, modifying the optimality criterion of the RL agent with a risk factor (Sato et al., 2001; Gaskett, 2003; Geibel and Wysotzki, 2005), or modifying the exploration process itself (Gehring and Precup, 2013; Garcia and Fernández, 2012). A typical property of the approaches in García and Fernández (2015) is that they *reduce* the risk of unsafe behavior, but there is no guarantee of risk *elimination*. However, one of the approaches shows an exciting concept to expand on, *Teacher Advising*. The concept is a way of altering the exploration process where a teacher with knowledge of the system can provide advice (e.g., safe actions) to the learner when either the learner or the teacher considers it necessary to prevent catastrophic situations (García and Fernández, 2015). This way of thinking is similar to the approach used in this thesis, namely predictive safety filtering, which acts as the *teacher* and filters the actions of the RL externally.

The combination of RL and traditional control theory is an active area of research (Xie et al., 2020; Paden et al., 2016; Kamthe and Deisenroth, 2018). MPC is an advanced control strategy known for its stability and constraint satisfaction guarantees, and is widely used in state-of-the-art applications (Gros and Schild, 2017; Hewing et al., 2020). An adaptation of MPC, called Predictive Safety Filter (PSF), has shown remarkable results (Wabersich and Zeilinger, 2018*b*) and could be the next step towards guarantees in safe RL. Instead of advising the agent, the PSF acts as an intermediary and filters actions that promote unsafe exploration. The PSF is optimizing for minimal intervention, based on its understanding of the underlying system, while still safekeeping the system. The modularity of it adds the ability of being compatible with any controller. The field is still novel and has seen few practical implementations.

Through adding a PSF, the RL agent no longer interacts freely with the environment, raising questions on training progress and performance with imposed exploration restrictions. This encouraged experimentation on a practical implementation. A FOWT was used as our dynamical system controlled by an RL agent in conjunction with a PSF.

1.1.1 State of the art

Floating offshore wind turbine

The NREL 5MW turbine (JM et al., 2009) has for a long time been the state-of-the-art reference wind turbine (RWT) used for research with its over 4500 citations according to Google Scholar. This turbine has been sufficient until recent years as the average turbine size for fixed-bottom offshore wind energy in Europe in 2018 was $6.8MW$. However, the turbine capacity has increased by 16% every year from 2014 to 2019 and continued to grow to 7.8 MW in 2019 (Europe, 2019). Higher demands for power generation and increasingly larger turbines in the industry led to the development of a new 15MW turbine in 2020. As explained by Gaertner et al. (2020): *GE will launch its 12-MW Haliade-X offshore turbine to the market in 2021 [...]. To be relevant now and in the coming years, a new reference wind turbine should leap ahead of the current generation of industrial wind turbines, but cannot leap so far that aggressive technology innovations are required. Therefore, a reference wind turbine above 10 MW, yet below 20 MW, is needed.* The IEA 15MW (Gaertner et al., 2020) is now becoming the new state of the art within wind turbine research. The turbine is developed as both a floating (Allen et al., 2020) and a monopile structure (Gaertner et al., 2020).

For control, the IEA 15MW reference turbine implements two proportional-integral (PI) controllers, one for the generator torque and another for the blade pitch angles. More advanced control methods for variable speed variable pitch (VSVP) turbines have been proposed. Examples are adaptive control based on Radial-Basis-Function Neural networks (Jafarnejadsani et al., 2012), and scheduled MPC (Kumar and Stol, 2009).

Reinforcement learning

RL is a rapidly developing field with the state of the art constantly evolving. There are a number of toolkits, frameworks, and libraries available for implementing, testing, and comparing RL algorithms in various applications. The OpenAI Gym toolkit (Brockman et al., 2016) has quickly become a state-of-the-art framework for RL applications and is widely used in research within this field. Its popularity reflects its ease of use, flexibility, and powerful capabilities. Stable Baselines3 (SB3) (Raffin et al., 2019) is a set of improved implementations of RL algorithms based on OpenAI Baselines. It has en easy to use interface and many state-of-the-art algorithms implemented, such as PPO (Schulman, Wolski, Dhariwal, Radford and Klimov, 2017), DDPG (Lillicrap et al., 2015), and TD3 (Fujimoto et al., 2018). SB3 is the next major version of Stable Baselines and introduces backend changes like a move from Tensorflow to Pytorch. SB3 is used in this thesis as it is considered a future-proof and intuitive approach to RL algorithm implementation.

Meyer (2020) presents a state-of-the-art approach to the continuous control application of autonomous vessels using the OpenAI Gym toolkit (Brockman et al., 2016), Stable Baselines (Hill et al., 2018), and the PPO RL algorithm (Schulman, Wolski, Dhariwal, Radford and Klimov, 2017). Teigen (2020) explains that PPO gave the best performance in this continuous control problem.

Safety filtering frameworks

To briefly outline a constantly evolving field of joint learning-based agents with control theory, we want to highlight two approaches. These two are highlighted due to their inherent modularity, in the sense of being compatible with any learning-based agent. Control Barrier Functions (CBF) are perhaps the most natural approach to filtering the agent's proposed actions. Loosely stated, this consists of explicitly calculating a function that describes a boundary between safe and unsafe. CBF was first introduced in Wieland and Allgöwer (2007), which is strongly tied to control Lyapunov function (CLF). However, the first formulation was stronger than necessary, and the more "modern" CBS was reintroduced in Ames et al. (2014). While there exist strong theoretical results with CBF, obtaining the explicit function has proven to be challenging, and while approximate solutions through sum-of-squares programming exist (Wang et al., 2018), this has been one of its main criticisms. However, more recently, Robey et al. (2020) proposed a learned control barrier function through expert demonstrations to address this issue. The reader should note that the CBF is often combined with a CLF to obtain a controller directly, more in line with Wieland and Allgöwer (2007). An example is Choi et al. (2020) where the model uncertainty is addressed with an RL agent. While this is also a constrained nonlinear control problem, similar to the task presented in the text, it differs to such a degree it will not be pursued with the scope of the current work.

PSF was first introduced in Wabersich and Zeilinger (2018*b*) and builds on Model Predictive Safety Certificates (MPSC) (Wabersich and Zeilinger, 2018*a*) proposed by the same authors. By defining the safe set implicitly through a learning-based MPC, the proposed method avoids explicitly calculating the CBF. The PSF presented in the original paper is model-free, thus avoiding the expert insight needed for more traditional MPC formulations. The authors further claim that it has favorable scalability and avoid over-conservatism compared to the general learning-based MPC (see Hewing et al. (2020) for an overview). The rigorous constraint satisfaction is inherently linked to traditional MPC (Mayne, 2014). It also addresses the problem of recursive constrain satisfaction often negated in safe RL approached previously mentioned.

1.2 Research objectives and research questions

1.2.1 Objectives

The primary objective of this work is to develop a framework for combining RL and PSF for safe AI in wind energy applications.

The secondary objectives are stated as:

- Evaluate the conditions under which an RL agent can control the state of a FOWT and optimize its power generation.
- Evaluate the feasibility and challenges of applying PSF to guarantee constraint satisfaction and safety during training and deployment of an RL agent through applying it on a FOWT system.
- Investigate the effect of a PSF on the training and learning of an RL agent in the application of a FOWT.

1.2.2 Research questions

To the best of our knowledge, there is currently no published work on combining RL and a PSF for control of a FOWT. To this end, the guiding questions governing the research can be stated as:

- Under which conditions is an RL agent able to successfully control the state of, and optimize power generation for, the FOWT without a PSF?

- To what degree is a PSF able to provide safety and constraint satisfaction guarantees for RL in practice for this application?
- How is the training progress and performance of the RL agent affected by the PSF?

1.3 Outline of the report

The thesis comprises of five chapters. **Chapter 2** explains the fundamental theories behind the work in this project including some mathematical background on dynamical systems, an introduction to wind turbines and RL as well as a presentation of the building blocks of a PSF; **Chapter 3** dissects the concrete methods and specifics of the setup used presenting our RL-PSF framework, the derivation for our wind turbine model and specifics of both the RL and the PSF implementation; **Chapter 4** presents the results and a discussion around them looking at both the successes and limitations of our work. The thesis is concluded in **Chapter 5**, where suggestions for future work are also presented.

Chapter 2

Theory

In this chapter, we will introduce some of the main building blocks of our thesis. We assume that the reader is familiar with both linear and nonlinear control theory. The reader should also understand general topics in optimization theory, such as convexity and gradient descent. Furthermore, while the theory on RL introduces the knowledge needed in this thesis, it is kept high-level, and the reader is directed to sources like Sutton and Barto (2018) for a more comprehensive read. The chapter's primary purposes are to give the reader the background to understand our work and establish notion and terminology.

2.1 Dynamical systems

We can express a dynamical system in the form of a system of ordinary differential equation (ODE) as

$$\frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}} = \begin{bmatrix} f_1(t, x_1, ..., x_{n_x}, u_1, ..., u_{n_u}, p_1, ...p_{n_p}) \\ f_2(t, x_1, ..., x_{n_x}, u_1, ..., u_{n_u}, p_1, ...p_{n_p}) \\ \vdots \\ f_n(t, x_1, ..., x_{n_x}, u_1, ..., u_{n_u}, p_1, ...p_{n_p}) \end{bmatrix} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{p}) \tag{2.1.1}$$

where $\mathbf{x} = [x_1, ..., x_{n_x}]^\intercal$ is the system state, $\mathbf{u} = [u_1, ..., u_{n_u}]^\intercal$ is the control input or actuation, and $\mathbf{p} = [p_1, ..., p_{n_p}]^\intercal$ contains external parameters or process disturbances. The system evolution in a discrete timestep $t + 1$ can be written as

$$\mathbf{x}(t + 1) = \mathbf{x}_{t+1} = \Phi(\tau_t, \mathbf{x}_t, \mathbf{u}_t, \mathbf{p}_t) \tag{2.1.2}$$

where $\Phi(\cdot)$ is the function mapping the dynamics from continuous time to discrete time and τ is discretization step length.

If the system is time invariant, it can be linearized around a point $\mathbf{z}_{lin} = [\mathbf{x}_{lin}, \mathbf{u}_{lin}]$. Removing $\mathbf{p}$ without loss of generality, we are left with the following equation

$$\dot{\mathbf{x}} = \nabla^\intercal \mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}}(\mathbf{x} - \mathbf{x}_{lin}) + \nabla^\intercal \mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}}(\mathbf{u} - \mathbf{u}_{lin}) + \mathbf{f}|_{\mathbf{z}_{lin}} \tag{2.1.3}$$

where $\nabla^\intercal \mathbf{f}_*$ is the gradient operator on $\mathbf{f}$ with respect to $*$. We can rewrite the system dynamics as Linear Time Invariant (LTI), obtaining the familiar expression

$$\dot{\mathbf{x}} = \nabla^\intercal \mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}}\mathbf{x} + \nabla^\intercal \mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}}\mathbf{u} + \nabla^\intercal \mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}}\mathbf{x}_{lin} + \nabla^\intercal \mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}}\mathbf{u}_{lin} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{b} \tag{2.1.4}$$

The system is not linear as it contains an affine term $\mathbf{b}$, however any affine function is linear in a higher dimension. By *lifting* or augmenting an affine system we can create a linear system through stating the system in higher dimension. For an affine continuous system the transformation can be seen below:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{b} \quad \leftrightarrow \quad \dot{\tilde{\mathbf{x}}} = \begin{bmatrix} \dot{\mathbf{x}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0} & 0 \end{bmatrix} \tilde{\mathbf{x}} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \mathbf{u} = \tilde{\mathbf{A}}\tilde{\mathbf{x}} + \tilde{\mathbf{B}}\mathbf{u} \tag{2.1.5}$$

To move the origin of the system to arbitrary point in space $\mathbf{z}_{c_0} = [\mathbf{x}_{c_0}^\intercal, \mathbf{u}_{c_0}^\intercal]^\intercal$, we redefine $\mathbf{z} = \mathbf{z}_c + \mathbf{z}_{c_0}$ and restate our system as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} = \mathbf{A}(\mathbf{x}_c + \mathbf{x}_{c_0}) + \mathbf{B}(\mathbf{u}_c + \mathbf{u}_{c_0}) = \mathbf{A}\mathbf{x}_c + \mathbf{B}\mathbf{u}_c + \mathbf{b} \tag{2.1.6}$$

2.1.1 Polytopic constraints

A polytope $\mathcal{X}$ refers to a convex set of points defined by a finite number of half spaces. The set can be unbounded in any direction, and does include its borders, viz

$$\mathcal{X} = \{\mathbf{x} | \mathbf{a}_i^\intercal \mathbf{x} + b_i \leq 0, \forall i \in \mathcal{H}\} \tag{2.1.7}$$

where the bounding hyperplanes have indices $\mathcal{H} = \{1, 2, ..., n\}$.

Also note that a polytope could be constrained by equality constraints $\mathbf{a}\mathbf{x} = b$, but this could always be expressed as two inequalities

$$\mathbf{a}\mathbf{x} \leq b \quad \cup \quad -\mathbf{a}\mathbf{x} \leq -b \tag{2.1.8}$$

If we construct a matrix $\mathbf{H}_x = [\mathbf{a}_1, ..., \mathbf{a}_n]$ and a vector $\mathbf{h}_x = [b_1, ..., b_n]$, we can express the polytopic constraint as a matrix of inequalities

$$\mathbf{H}_x\mathbf{x} \leq \mathbf{h}_x \quad \leftrightarrow \quad \mathrm{row}_i(\mathbf{H}_x) \leq \mathrm{row}_i(\mathbf{h}_x) \tag{2.1.9}$$

As an example, a square $\mathcal{D}$ centered at the origin with side length 2 can be expressed as

$$\mathcal{D} \quad \leftrightarrow \quad \begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{2.1.10}$$

With polytopic constraints $\mathcal{X}, \mathcal{U}$ on state and input respectively, we can still move our system to an arbitrary point $\mathbf{z}_{c0}$ with Equation 2.1.6. However, we also need to move our constraints as follows,

$$\mathbf{H_z}\mathbf{z} < \mathbf{h_z} \quad \leftrightarrow \quad \mathbf{H_z}(\mathbf{z}_c + +\mathbf{z}_{c0}) < \mathbf{h_z} \quad \leftrightarrow \quad \mathbf{H_z}\mathbf{z}_c < \mathbf{h_z} - \mathbf{H_z}\mathbf{z}_{c_0} \tag{2.1.11}$$

2.1.2 Runge-Kutta methods

There exist wide range of methods to evaluate the discrete system evolution. We heavily rely on the numerical method of Runga-Kutta (RK) in its explicit form. Its general form with ν steps and one state, input and external parameter is

$$\begin{aligned} x_{t+1} &= x_t \sum_{i=1}^{\nu} b_j f(t + c_j \tau_t, \xi_j) + \tau_t(u+p) \\ \xi_j &= x_t + \tau_t \sum_{i=1}^{\nu-1} a_{\nu,i} f(t + c_j \tau_t, \xi_i) \end{aligned} \tag{2.1.12}$$

where u_t and p_t are kept constant through the step length τ_t. The constants $a_{\nu,i}, b_j, c_i$ can be rearranged into a Butcher tableau, as seen in Table 2.1.1. The elements of the matrix $\mathbf{A}_{RK}$ are the constants $a_{\nu,i}$, and the vector $\mathbf{c}_{RK}, \mathbf{b}_{RK}$ consists of b_j and c_i respectively.

$\mathbf{c}_{RK}$	$\mathbf{A}_{RK}$
	$\mathbf{b}_{RK}^\intercal$

Table 2.1.1: Butcher tableau in matrix form.

For explicit RK-methods, we can investigate the linear stability of the method through the stability function $R_E(\mu)$ (Egeland and Gravdahl, 2002),

$$R_E(\mu) = R_E(\lambda\tau_t) = \det\left[\mathbf{I} + \lambda\tau_t(\mathbf{A}_{RK} + \mathbf{1}\mathbf{b}_{RK}^\intercal)\right] \tag{2.1.13}$$

where λ represents the eigenvalues of the linear system. The discretization is stable if $|R_E(\mu)| < 1$. For RK4 this expression simply becomes

$$R_E(\mu) = 1 + \mu + \frac{1}{2}\mu^2 + \frac{1}{6}\mu^3 + \frac{1}{24}\mu^4 \tag{2.1.14}$$

2.1.3 Semi-definite programs

We define the matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ as positive semi-definite, if it is symmetric and $\mathbf{x}^\intercal \mathbf{M}\mathbf{x} \geq 0$ for all $\mathbf{x}$. The following statements are equivalent

$$\mathbf{x}^\intercal \mathbf{M}\mathbf{x} \geq 0 \quad \leftrightarrow \quad \mathbf{M} \succeq 0 \tag{2.1.15}$$

Definiteness plays a large role in nonlinear control and optimization, in this space given rise to a Semi-Definite Programs (SDP), which is defined as

$$\min \mathbf{c}^\intercal \mathbf{x} \quad s.t \quad \mathbf{F}(\mathbf{x}) \succeq 0 \tag{2.1.16}$$

where

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}_0 + \sum_{i=1}^{n} \mathbf{F}(x_i) \tag{2.1.17}$$

The inequality $\mathbf{F}(\mathbf{x}) \succeq 0$ is a Linear Matrix Inequality (LMI). Note that solvers for large-scale SDP, and its generalization Log-Log Convex Programs (LLCP), exists (Diamond and Boyd, 2016; Löfberg, 2004). For a comprehensive treatment we refer the reader to Boyd et al. (1994) and Agrawal et al. (2019).

2.2 Floating offshore wind turbines

In this section, we introduce terminology for and explain the concept of a wind turbine. The specifications and construction of wind turbines vary significantly. As a consequence, we limit our explanation to a high-level understanding of the fundamentals.

A wind turbine is a device that converts the kinetic energy of wind to electrical energy. There are two main categories of wind turbines, horizontal and vertical, with horizontal being the most common. The turbine can be installed with several methods depending on the use case. As presented in section 1.1, the focus of this thesis will be offshore installations with the IEA 15MW turbine as a basis. There are three main floating offshore platform types; Tension Leg Platform, Semi-Submersible, and Spar, with the IEA 15MW being a Semi-Submersible. The differences are mainly related to how the platform is moored and its floating characteristics. The platform types will not be a focus in this report, as severe simplifications to the platform model are made, rendering the platform mentioned above types irrelevant. On top of the base structure is a tower that holds the remaining components. The base and tower as a whole will be referred to as the *platform* in this thesis.

The power generation of a wind turbine comes from its generator, placed in the *nacelle* on top of the platform. The nacelle is then attached to a rotor, commonly with three blades. The blades turn the wind's translational energy to rotational energy for the generator, which turns it into electric energy by adding a counter-torque. We refer to the blades and generator as the *rotor*. The whole structure, the platform and the rotor in combination, make up what we call the *turbine*.

Assuming a direct-drive generator, which is consistent with the IEA 15MW, the rotational velocity of the rotor will be the same as that of the generator, i.e., $\Omega_{gen} = \Omega$. The generated power P_{gen} and generator torque Q_{gen} then follow the equations

$$\begin{aligned} P_{gen} &= J_r \Omega \dot{\Omega} = Q_{gen} \Omega \\ Q_{gen} &= \frac{P_{gen}}{\Omega}, \quad \Omega > 0 \end{aligned} \tag{2.2.1}$$

where Ω is the rotor angular velocity, $\dot{\Omega}$ is the rotor acceleration, and J_r is the inertia of the rotor.

Variable speed variable pitch (VSVP) turbines

The IEA 15MW is a VSVP turbine with control possibilities for both blade pitch and generator torque. The blade pitch angle controls how much of the wind's energy is converted into rotational torque through the principle of lift and drag. Imagine a straight blade directed directly towards the wind. This configuration will lead to no rotational energy being created, while angling the blade will create a force pushing sideways on the blade (lift), leading to rotational torque on the rotor. The blade pitch is useful for maintaining a constant rotor velocity. A constant blade pitch angle would lead to rotor velocity increasing with wind speed, possibly damaging internal parts at high velocities. The wind also creates a force pushing backwards on the blades (drag), which acts as an axial force at the nacelle, rotating and pushing the structure away from the vertical position. Adjustment of the generator torque is used to control both the rotor velocity and the power generation. One use case of this is for maintaining the rotor velocity at lower wind speeds, where the generator could reduce its counter-torque, letting the rotor spin more freely. This naturally comes at the cost of less power being generated, as power and torque are directly proportional at constant rotor speeds, see Equation 2.2.1.

Wind spectrum

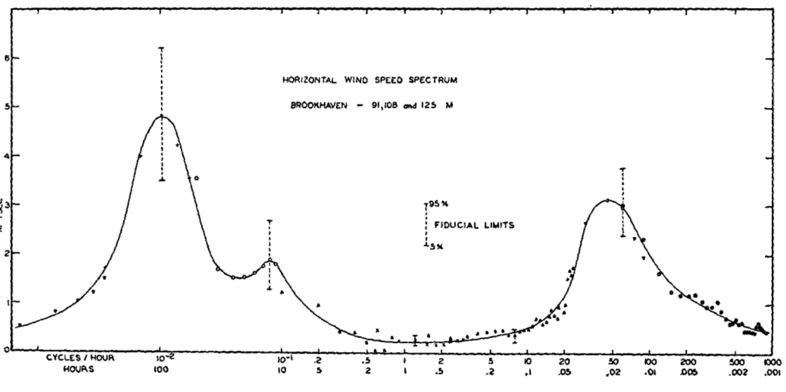


Figure 2.2.1: Power spectrum of horizontal wind speed measured at Brookhaven National Laboratory, from Van der Hoven (1957).

A power spectrum of wind speed from Brookhaven National Laboratory can be seen in Figure 2.2.1. This shows two main peaks, one representing slow-varying long-term variations in the weather systems at 10^{-2} cycles/hour, and one peak at higher frequencies representing faster fluctuations in wind speed at 60 cycles/hour.

Maximum power point tracking

Due to the instantaneous changing nature of the wind, it is desirable to determine the optimal generator speed that ensures maximum energy yield. Thus, it is essential to include a controller that can track the maximum peak regardless of wind speed (Abdullah et al., 2012).

Maximum Power Point Tracking (MPPT) is a technique commonly used in wind turbines and photovoltaic solar systems to maximize power extraction under all conditions. In the case of wind turbines it is common to use this to control the rotor speed using blade pitch angle.

Engineering model

Pedersen (2017) presents a simplified engineering model for the rotor system of a wind turbine. The model is presented further in section 3.1, along with our adaptation of it. The reader is directed to the original paper for a comprehensive derivation.

2.3 Deep reinforcement learning

In this section, we introduce the relevant theory within the field of Deep RL. For a more comprehensive read, the reader is directed to Sutton and Barto (2018) and Li (2018). This section is also largely based on Teigen (2020), one of the authors' specialization project.

Within the realm of Artificial Intelligence (AI), Machine Learning (ML) has been the most popular approach in recent years. We usually categorize ML by supervised, unsupervised, and RL. In supervised learning, the desired output needs to be known in order to train the model, i.e., labeled data is needed. This ML technique can be used in applications like regression and classification. Unsupervised learning seeks to find patterns and relevant information within unlabeled data. RL takes an altogether different approach that uses the principle of trial and error to extract an optimal strategy to solve a problem. RL can also be thought of as semi-supervised learning where the reward is a kind of time-delayed label. In this section, we will dive deeper into the topic of Deep RL. The term *deep* refers to the use of deep neural networks in ML approaches. This articture can be incorporated into all of the categories mentioned above.

On the one hand, there are value-based methods based on Temporal Difference (TD) Learning, while on the other, there are policy gradient methods based on policy optimization. The algorithm used in this project (PPO) adopts the actor-critic framework, which combines functionality from both of these methods and has shown excellent results for continuous control applications (Meyer, 2020; Schulman, Wolski, Dhariwal, Radford and Klimov, 2017). The upcoming parts of this section will introduce the RL framework, explain the two main approaches to RL algorithms, and eventually lead to PPO and the actor-critic framework and why it is used.

2.3.1 The reinforcement learning framework

The two main components in RL are the environment and the agent. These entities interact through actions, rewards, and states or observations. From a high-level perspective, the flow can be explained as follows: The agent performs an action a_t on the environment, which changes the state from s_t to s_{t+1}. The new state s_{t+1}, or a partial observation of it, is received by the agent along with a reward r_t indicating how good the action was. This reward is then used to improve the policy $\pi(a_t|s_t)$, which is a set of rules that the agent follows to decide which action to take next. A graphical representation of this flow can be found in Figure 2.3.1. A common way of implementing this is that the RL agent repeats this interaction for multiple timesteps until an end condition is met. This is defined as an *episode*. The environment is then reset, and the process is repeated for many episodes. The policy can be improved online at each timestep, sparsely at the end of each episode, or any other custom adaptation of this depending on the algorithm.

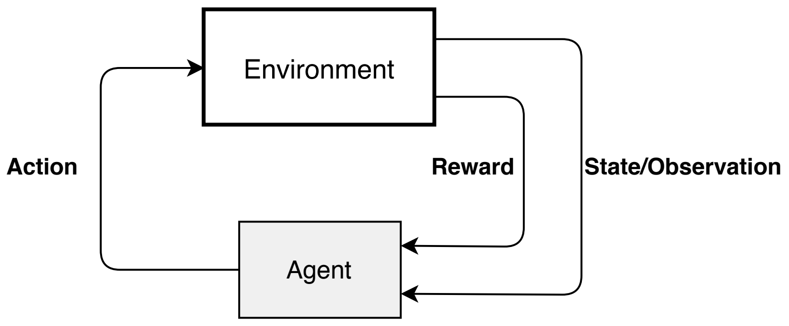


Figure 2.3.1: Overview of the RL framework.

An assumption made in most RL algorithms is that the problem can be formulated as a Markov Decision Process (MDP). A key attribute of this is that the future states depend only on the current state and action, not the past. The *model* of an MDP is defined by a transition function T giving the probability of moving to a state s' given a state s and an action a, and a reward function R giving the reward. If the model of the MDP is known, traditional optimization techniques can be used to find the optimal policy. This is often not the case, and an approach like RL is needed to solve this. Model-based RL strives to estimate this MDP model while the model-free approach focuses on the policy, or control strategy, itself.

The goal for the RL algorithm is to find the optimal policy, and it does this by maximizing the cumulative future reward at each timestep t

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.3.1}$$

where r_t is the reward at time t. R_t also contains a discount factor $\gamma \in (0, 1]$ which dictates how much the agent cares about future rewards. A γ of 1 gives equal weight to all rewards, regardless of temporal conditions, while a smaller γ results in the short term rewards getting weighted higher than the long term. A disount factor strictly less then 1 is usually preferred because, in general, a good action (thus a large reward) is worth more now than far into the future.

Exploration vs exploitation

An important consideration in RL is the trade-off between exploration and exploitation. On the one hand, we need the agent to explore the environment and evaluate as many different strategies as possible to stop it from converging to a local optimum with sub-optimal performance. On the other hand, we want the agent to exploit the information in the current policy to avoid completely random behavior. So how much of this exploration should the agent do, and how should it do it? This is a big question within RL and comes down to tuning for the respective algorithm and application.

2.3.2 Value-based methods

Value-based methods are based around what is known as an action-value function, $Q(s, a)$. The action-value function estimates how good it is for an agent to be in a given state s and perform a given action a. Formally, this is the expected future reward for a given state-action pair. The algorithms then sample the MDP to gather statistical knowledge about the unknown model. In this way, the RL framework is used to estimate the complete action-value function and use it to make an optimal policy, i.e. choose the optimal action given a state.

Although value-based methods have some great properties and features like good sample efficiency and fast learning, an important thing to note is that they, in general, do not scale well to continuous action spaces. Imagine the action-value function $Q(s, a)$ as a table of values for states and actions. There are infinitely many possible action values for a continuous action space, which in turn means that the table will get infinitely large. This intractability makes it difficult and computationally expensive to calculate and find the maximum thereof. Due to this weakness, value-based methods are often combined with policy-based methods in what we call an actor-critic framework when applied to problems with continuous state and action spaces.

2.3.3 Policy gradient methods

While value-based methods optimize the policy through a value function, policy-based methods have a different and more direct approach. The policy $\pi(a|s;\mathbf{W})$, parameterized with parameters $\mathbf{W}$, is optimized directly through gradient ascent on the expected reward. The parameters $\mathbf{W}$ can, for instance, be the weights of a deep neural network.

This approach comes with several advantages. Earlier, we mentioned that value-based methods scale poorly to growing action spaces. This problem is not as prominent in policy-based methods because instead of computing learned probabilities for each of the actions; it learns statistics of the probability distribution (Sutton and Barto, 2018). In addition, the policy itself may be a more straightforward function to approximate than the action-value function. Policy gradient methods also have the ability to find stochastic optimal policies, something that action-value methods do not have (Sutton and Barto, 2018), and policy parameterization is a good way to introduce prior knowledge of the problem (Sutton and Barto, 2018), which is very useful from an engineering perspective. There are, of course, drawbacks to policy gradient methods, with the most significant being sample inefficiency and high variance.

2.3.4 Actor-Critic methods

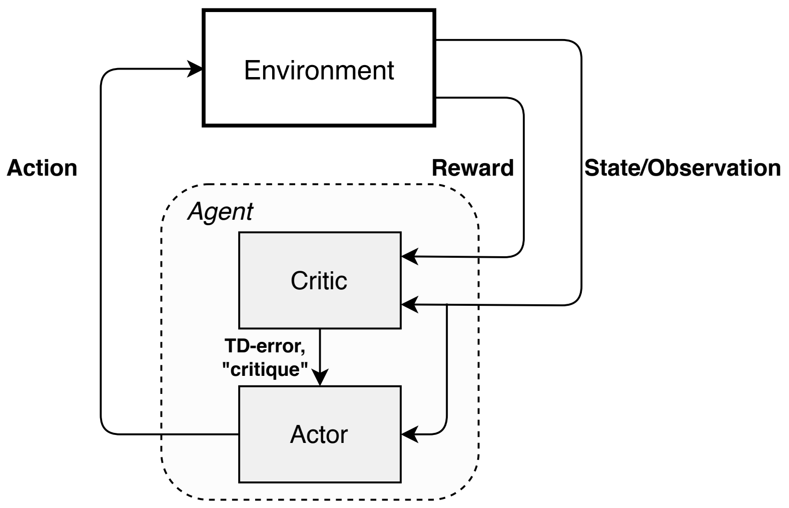


Figure 2.3.2: Actor-Critic framework for RL.

The most desirable approach would be to combine the advantages of both value and policy-based methods or at least mitigate some of the drawbacks of one by leveraging the other. The actor-critic method does precisely this and can be seen as a kind of hybrid approach. As seen in Figure 2.3.2 it uses both a parameterized policy in the actor and a value function in the critic. The actor calculates which action to take in a given state while the critic evaluates the action taken and gives a *critique*, in the form of an error based on the value function, to the actor in order to improve the policy further. This brings the benefits from value-based methods, like better sample efficiency, together with the advantages of policy-based methods, like the ability to handle large and continuous state and action spaces.

Proximal Policy Optimization

Proximal Policy Optimization(PPO) is a model-free RL algorithm that uses an actor-critic architecture. PPO is based on the principle of a trust region, i.e., improving the policy as much as possible without going too far from where we are and breaking the policy. It implements this in a simple and computationally less demanding way compared to other trust-region methods like TRPO (Schulman, Levine, Moritz, Jordan and Abbeel, 2017). The original paper (Schulman, Wolski, Dhariwal, Radford and Klimov, 2017) presents results where PPO outperforms both A2C and A2C + Trust Region (Wu et al., 2017) in several continuous control tasks. It has also shown great results in Meyer (2020) and was deemed the best performing algorithm for control of autonomous vessels in Teigen (2020).

2.4 Building blocks of the predictive safety filter

PSF was first introduced in Wabersich and Zeilinger (2018*b*) and extended in Wabersich and Zeilinger (2021*a*) to accommodate learning-based system models with uncertainty. The method is based on trajectory optimization and can be seen as a relaxed MPC formulation. In this section, the theoretical foundation of the PSF's main building blocks will be introduced before returning to our PSF implementation in chapter 3.

2.4.1 Terminal set

We define a set as a safe set $\mathcal{S}$ if does not violate any state constraints $\mathcal{X}$ after entering the set $\mathcal{S}$. Through this definition, the common notion of a control invariant set $\mathcal{C} \subseteq \mathcal{X}$ is with safe set. A control invariant set is described as

$$\mathbf{x}_t \in \mathcal{C} \quad \rightarrow \quad \exists \mathbf{u}_t \quad s.t \quad \Phi(\mathbf{x}_t, \mathbf{u}_t) \in \mathcal{C}, \quad \forall t \in \mathbb{R}^+ \tag{2.4.1}$$

The maximum control invariant set provides largest safe set, but can be very difficult to compute. However, there exist several methods of to obtain a smaller control invariant set.

Linear feedback controller

A well-developed method is to use a state feedback controller $\mathbf{u}_t = \pi(\mathbf{x}_t)$ as a control policy. When the system is linear, this becomes a linear feedback controller

$\mathbf{u}_t = \mathbf{K}\mathbf{x}_t$. With Lyapunov analysis, excellently described in Khalil (2015, chap 5), we can formulate the necessary condition to obtain the set $\mathcal{C}$ through the feedback controller. The function $V(x) = \mathbf{x}^\intercal\mathbf{P}\mathbf{x} > 0$ serves as the Lyapunov candidate when considering both the continuous and discrete case.

In **continuous-time formulations** , the linear system is stable if the time derivative is negative for all $\mathbf{x}$, viz.

$$\frac{dV(\mathbf{x})}{dt} = \dot{\mathbf{x}}^\intercal\mathbf{P}\mathbf{x} + \mathbf{x}^\intercal\mathbf{P}\dot{\mathbf{x}} < 0 \tag{2.4.2}$$

$$(\mathbf{A}+\mathbf{B}\mathbf{K})^\intercal\mathbf{P} + \mathbf{P}(\mathbf{A}+\mathbf{B}\mathbf{K}) \prec 0 \tag{2.4.3}$$

To solve the equation above, a semi-definite optimization scheme can be deployed. However, most solvers only accept linear matrices, since both $\mathbf{P}$ and $\mathbf{K}$ are unknown matrices in the system, the problem is bi-linear. Using congruence transformation (Boyd et al., 1994), i.e. pre- and post multiplying with $\mathbf{Q} = \mathbf{P}^{-1}$, the problem can be restated as

$$\mathbf{Q}\mathbf{A}^\intercal + \mathbf{A}\mathbf{Q} + \mathbf{Q}\mathbf{K}^\intercal\mathbf{B}^\intercal + \mathbf{B}\mathbf{K}\mathbf{Q} \prec 0 \tag{2.4.4}$$

After defining the variable $\mathbf{L} = \mathbf{K}\mathbf{Q}$, we arrive at the final linear semi-definite expression

$$\mathbf{Q}\mathbf{A}^\intercal + \mathbf{A}\mathbf{Q} + \mathbf{L}^\intercal\mathbf{B}^\intercal + \mathbf{B}\mathbf{L} \prec 0 \tag{2.4.5}$$

The feedback gain $\mathbf{K}$ can always be recuperated from $\mathbf{L}$ since $\mathbf{P}$ hence $\mathbf{Q}$ is positive definite and hence full rank.

In **discrete time formulation**, we relax the negative definiteness of the Luapunov derivative. Instead, we require that at each timestep the Luapunov function is decreasing, viz. $V(x_{t+1}) < V(x_t)$. In matrix form that is

$$\mathbf{x}_{t+1}^\intercal\mathbf{P}\mathbf{x}_{t+1} < \mathbf{x}_t^\intercal\mathbf{P}\mathbf{x}_t \tag{2.4.6}$$

$$\mathbf{x}_k^\intercal(\mathbf{A}+\mathbf{B}\mathbf{K})^\intercal\mathbf{P}(\mathbf{A}+\mathbf{B}\mathbf{K})\mathbf{x}_t < \mathbf{x}_t^\intercal\mathbf{P}\mathbf{x}_t \tag{2.4.7}$$

$$(\mathbf{A}+\mathbf{B}\mathbf{K})^\intercal\mathbf{P}(\mathbf{A}+\mathbf{B}\mathbf{K}) - \mathbf{P} \prec 0 \tag{2.4.8}$$

Using Schur's complement (see Zhang (2006)), we can expand the equations to

$$\begin{bmatrix} \mathbf{P} & (\mathbf{A}+\mathbf{B}\mathbf{K})^\intercal\mathbf{P} \\ \mathbf{P}(\mathbf{A}+\mathbf{B}\mathbf{K}) & \mathbf{P} \end{bmatrix} \prec 0 \tag{2.4.9}$$

Again the formulation is bi-linear, but using the congruence transform and a change of variables we obtain our final expression in the discrete case:

$$\begin{bmatrix} \mathbf{P}^{-1} & \mathbf{P}^{-1}(\mathbf{A}+\mathbf{B}\mathbf{K})^\intercal \\ \mathbf{P}^{-1}(\mathbf{A}+\mathbf{B}\mathbf{K}) & (\mathbf{A}+\mathbf{B}\mathbf{K})\mathbf{P}^{-1} \end{bmatrix} \succ 0 \tag{2.4.10}$$

$$\begin{bmatrix} \mathbf{P}^{-1} & \mathbf{P}^{-1}\mathbf{A}^\intercal + \mathbf{P}^{-1}\mathbf{K}^\intercal\mathbf{B}^\intercal \\ \mathbf{A}\mathbf{P}^{-1} + \mathbf{B}\mathbf{K}\mathbf{P}^{-1} & \mathbf{P}^{-1} \end{bmatrix} \succ 0 \tag{2.4.11}$$

$$\begin{bmatrix} \mathbf{E} & \mathbf{E}\mathbf{A}^\intercal + \mathbf{Y}^\intercal\mathbf{B}^\intercal \\ \mathbf{A}\mathbf{E} + \mathbf{B}\mathbf{Y} & \mathbf{E} \end{bmatrix} \succ 0 \tag{2.4.12}$$

Ellipsoidal maximization

While any $\mathbf{P}$ and $\mathbf{K}$ that satisfies the constraints above stabilizes the system in an unbounded case, a natural extension is to find the *largest* set where we can satisfy the constraints. This is done through noting that $\mathbf{x}^\intercal\mathbf{P}\mathbf{x}$ spans an ellipsoid

$$\mathcal{E}(\mathbf{P}) = \{\mathbf{x}|\mathbf{x}^\intercal\mathbf{P}\mathbf{x} \leq 1\} \in \mathbb{R}^{n_x} \tag{2.4.13}$$

The eigenvalues of $\mathbf{P}$ are the squared reciprocal of the semi-axis spanning the ellipse, hence the volume can be described as $\frac{4}{3}\pi\prod_{n_x}\lambda_i^{\frac{1}{2}}$. Through maximizing the determinants of $\mathbf{P}^{-1} = \mathbf{E}$, we maximize the size of the ellipse $\mathcal{E}(\mathbf{P})$ (Boyd et al., 1994).

However, the ellipsoid maximization does not encompass the state and system constraints. Given that state and input are independent polytopic constraints, as such

$$\mathcal{X} = \{\mathbf{x}|\mathbf{H}_x\mathbf{x} \leq \mathbf{b}_x\} \qquad \mathcal{U} = \{\mathbf{u}|\mathbf{H}_u\mathbf{u} \leq \mathbf{b}_u\} \tag{2.4.14}$$

we can constrain the ellipse to adhere with the following inequalities (Wabersich and Zeilinger, 2018*b*):

$$\begin{bmatrix} \mathrm{row}_i(\mathbf{b}_x^2) & \mathrm{row}_i(\mathbf{H}_x)\mathbf{E} \\ (\mathrm{row}_i(\mathbf{H}_x)\mathbf{E})^\intercal & \mathbf{E} \end{bmatrix} \succeq 0, \forall i \tag{2.4.15a}$$

$$\begin{bmatrix} \mathrm{row}_j(\mathbf{b}_u)^2 & \mathrm{row}_j(\mathbf{H}_u)\mathbf{Y} \\ (\mathrm{row}_j(\mathbf{H}_u)\mathbf{Y})^\intercal & \mathbf{E} \end{bmatrix} \succeq 0, \forall j \tag{2.4.15b}$$

where i and j represents the row vector in the polytopic constraints, i.e. each half-space constraint. The problem is an LLCP, making it applicable to theory and solvers addressed in subsection 2.1.3.

Robust stability under polytopic uncertainty

Consider the uncertain control system $\dot{\mathbf{x}} = \mathbf{A}(\boldsymbol{\delta})\mathbf{x} + \mathbf{B}(\boldsymbol{\delta})\mathbf{u}$, where $\boldsymbol{\delta}$ is an uncertainty parameter. Given that $\forall\boldsymbol{\delta} \in \Delta$, where Δ is a polytopic uncertainty set with vertices $\Delta_g = \{\delta^1, ..., \delta^N\}$. The system is quadratically stable for all $\boldsymbol{\delta}$ if the following constraints hold (Scherer and Weiland, 2000):

$$\mathbf{P} \succ 0 \quad \mathbf{Q}\mathbf{A}(\delta^i)^\intercal + \mathbf{A}(\delta^i)\mathbf{Q} + \mathbf{Q}\mathbf{K}^\intercal\mathbf{B}(\delta^i)^\intercal + \mathbf{B}(\delta^i)\mathbf{K}\mathbf{Q} \prec 0, \forall i \in \mathbb{I}_{1,N} \tag{2.4.16}$$

Less formally, if we find a $\mathbf{K}$ which stabilizes all the "extremes" of the system, we know that the entire system is stable. We can once again can use the notion of invariance, where the set $\mathcal{C}^{\Delta} \subseteq \mathcal{X}$ is a robust control invariant set, define as the following

$$\mathbf{x}_t \in \mathcal{C}^{\Delta} \quad \rightarrow \quad \exists \mathbf{u}_t \quad s.t \quad \Phi(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\delta}) \in \mathcal{C}^{\Delta}, \quad \forall \boldsymbol{\delta} \in \Delta, \quad \forall t \in \mathbb{R}^+ \tag{2.4.17}$$

2.4.2 Model predictive control

MPC is a dynamic optimization technique that is the de-facto standard within advanced control methods in the process industries (Qin and Badgwell, 1997; Mehrizi-Sani, 2017; Johansen, 2011), due to its ability to handle constraints and multivariate systems.

The main idea is to optimize the state trajectory for a given horizon N but only apply the first control input. After receiving a system update, the optimization problem is solved again for the new states. The state trajectory optimization is done through minimizing a cost function $\mathcal{J}$ constrained by the state trajectory $\Phi(\cdot)$, state constraints $\mathcal{X}$, and input constraints $\mathcal{U}$ a combination thereof. While a continuous-time formulation is possible (Wang, 2001), we limit our discussion to the discrete case. The reader should note that we simplify the notation, where $\mathbf{x}_k$ is really $\mathbf{x}_{k|t}$, that is the state $\mathbf{x}$ at optimization timestep k given the system time t.

Linear model predictive control

If the system is linear with polytopic constraints, a common formulation for linear MPC is

$$\begin{aligned}
&\min_{\mathbf{X},\mathbf{U}} \sum_{k=0}^{N} \mathbf{x}_{k+1}^{\intercal}\mathbf{Q}\mathbf{x}_{k+1} + \mathbf{u}_k^{\intercal}\mathbf{R}\mathbf{u}_k + \mathbf{d}_x\mathbf{x} + \mathbf{d}_u\mathbf{u} && (2.4.18a)\\
&s.t. \\
&\mathbf{x}_{k=0} = \mathbf{x}_t && (2.4.18b)\\
&\mathbf{x}_{k+1} = \mathbf{A}_k\mathbf{x}_k + \mathbf{B}_k\mathbf{u}_k \qquad \forall k \in \mathbb{I}_{\leq N} && (2.4.18c)\\
&\mathbf{H}_x\mathbf{x}_k \leq \mathbf{h}_x && (2.4.18d)\\
&\mathbf{H}_u\mathbf{u}_k \leq \mathbf{h}_u && (2.4.18e)\\
&\Delta\mathbf{u}_k \leq \mathbf{h}_{\Delta u} && (2.4.18f)
\end{aligned}$$

where $\Delta\mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}$ is a rate constraint.$\mathbf{Q}$, $\mathbf{R}$, $\mathbf{d}_x$, and $\mathbf{d}_u$ are costs associated with state and input. When $\mathbf{Q} \succcurlyeq 0, \mathbf{R} \succ 0$ the problem is convex.

Nonlinear model predictive control

Even though linear MPC has been the most widespread, systems that exhibit highly nonlinear properties may not lend themselves to linear MPC. Linearization around an operating point could prove beneficial, but formulations with, for example, long horizon could suffer dramatically. While there exist several ways to implement the optimal nonlinear control problem (Von Stryk, 1993; Allgöwer and Zheng, 2012), we present the numerical optimization scheme of direct multiple shooting (Leineweber et al., 2003)

$$\min_{\mathbf{X},\mathbf{U}} \sum_{k=0}^{N} \mathcal{J}(\tau_k, \mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k) \tag{2.4.19a}$$

$$s.t.$$

$$\mathbf{x}_{k=0} = \mathbf{x}_t \tag{2.4.19b}$$

$$\mathbf{x}_{k+1} = \Phi(\tau_k, \mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k) \qquad \forall k \in \mathbb{I}_{\leq N} \tag{2.4.19c}$$

$$0 \geq h(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k) \tag{2.4.19d}$$

where $h(\cdot)$ is an arbitrary constraining function and N is number of time steps considered (i.e. length of the horizon). $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_{N+1}]$ is the state sequence, $\mathbf{U} = [\mathbf{u}_0, ..., \mathbf{u}_N]$ is the input sequence and $\mathbf{p}$ is the external parameters unaffected by state and input. The formulation is not the most traditional, but is modified to suite the problem at hand.

The term *direct* refers to fixing the time grid before optimization, $\tau_k = [\tau_0, ..., \tau_N]$. In contrast to single shooting, *multiple shooting* retains the state sequence as an optimization variable, leading to a finite set of nonlinear algebraic equations which has to be solved simultaneously. As a result, this approach leads to a higher number of variables and constraints, causing the problem to become "bigger". This is often accepted due to ease of constraints and cost formulation, in addition to advantageous numerical properties (Johansen, 2011).

Model predictive control properties

Most of the success of MPC is commonly attributed to the re-optimization of the control problem at each timestep. However, this has also been its theoretical crux. Before the 2000s, proofs on stability, robustness, and optimality were sparse and primarily empirical. The highly acclaimed paper Mayne et al. (2000) can be seen as a landmark, establishing proofs on stability and optimality. Another problem with the re-optimization is the computational complexity. While solvers are getting faster, this is still a primary concern (Pedersen, 2017, Chap 1). While optimality and robustness are paramount in many cases, we refer the reader to Magni et al. (2009) due to the scope of this text. However, a short a outline of stability, in the sense of recursive feasibility, will be given due to its central role in this thesis. For a comprehensive introduction to a wide array of MPC-related topics we refer the reader to Mayne (2014) and Johansen (2011).

We will present the nominal case (without the external parameter $\mathbf{p}$), with the notion and presence of a terminal set $\mathcal{T} \subseteq \mathcal{S}$. For robust feasibility, we refer the reader to Mayne et al. (2011), and proofs without the terminal set can be found in Grüne (2012).

While several methods exist to obtain the terminal set, a common approach is to use dynamic programming to solve the Bellman equation. This equation is also commonly seen in value-based RL methods. The control policy $\pi(\cdot)$ obtained is associated with a set, where $\pi(\cdot)$ stabilizes the system for all time. It also has the benefit of a necessary condition of optimality. However, the recursive stability proof only relies on *any* control invariant set $\mathcal{C}$, e.g., the ellipsoidal set described in subsection 2.4.1.

Consider the case where a terminal set $\mathcal{T}$ exists. Let $\mathcal{T}$ act as a constraint on the MPC on the last step in the horizon $\mathbf{x}_{k+N+1} \in \mathcal{T}$. Assuming that we find a feasible control sequence $\mathbf{U}_{0,N} = [\mathbf{u}_0, \mathbf{u}_1 ..., \mathbf{u}_N]$ at timestep t_0 that drives the system to the terminal set, we know that the system will stay in the terminal set. At the next timestep $t > t_0$, we can deploy the previous sequence shifted forward $\mathbf{U}_{1,N}$, thus entering the terminal set.

Chapter 3

Method and setup

In this chapter, we explain the specifics of how the work was conducted. The chapter also includes an explanation of the training framework and a derivation of the model used. We also present the specifics of our experimental setup to give detailed insight and enable reproducibility.

3.1 Methodology

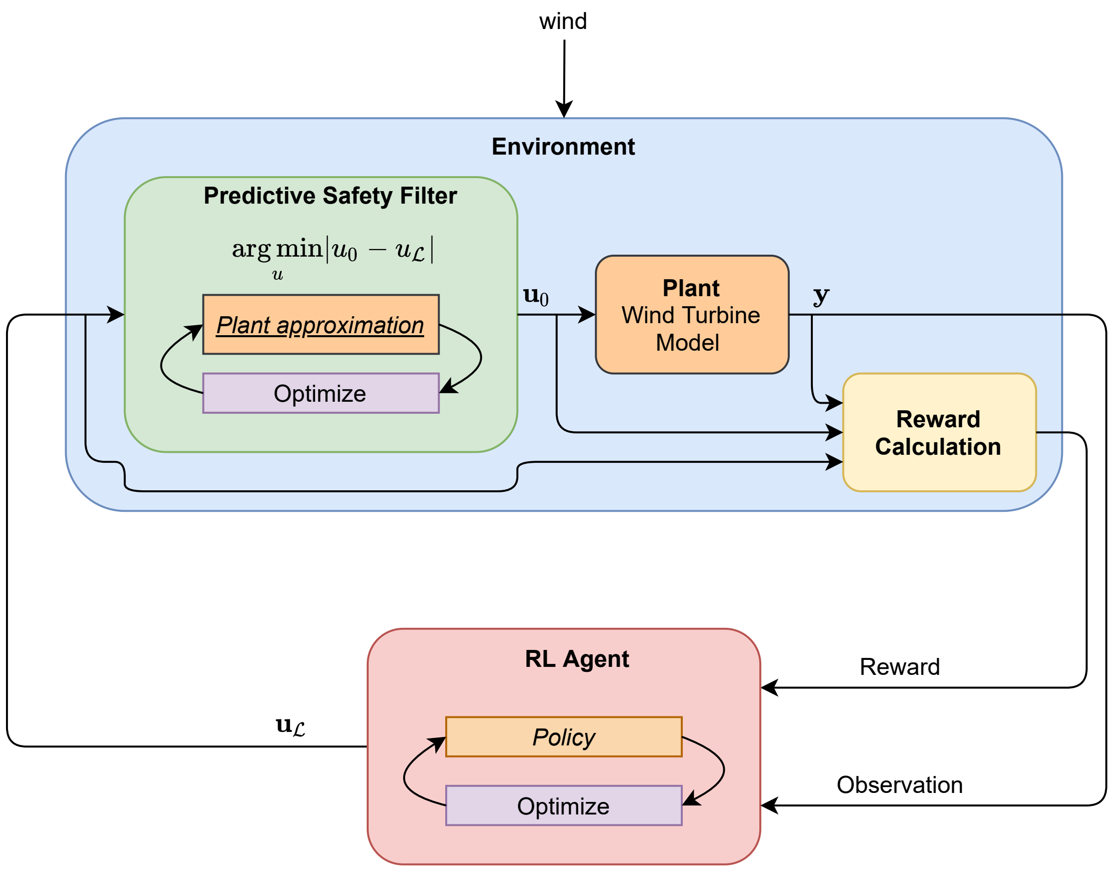


Figure 3.1.1: RL-PSF framework overview.

3.1.1 Reinforcement learning - predictive safety filter framework overview

Figure 3.1.1 shows an overview of our implemented framework. It consists of an RL agent and an environment composed of a PSF and a plant. The RL agent's proposed control input $\mathbf{u}_{\mathcal{L}}$ is filtered through the PSF, before sending the safe action $\mathbf{u}_0$ to the underlying plant. During training, a reward is calculated based on the resulting system state. Together with the observation $\mathbf{y}$, the reward is used by the RL agent to optimize its policy. The PSF calculates a backup trajectory, and alters the proposed control input in the case of future constraint violations. This leads to the agent having to learn the *combined* dynamics of the plant and PSF. Through this process the framework facilitates safe training and deployment of RL agents on any system.

The framework is highly modular, where each component can be improved upon with minimal dependency on the other modules. Relevant examples are replacing the plant and plant approximation for more accurate models, improving the reward function, migration towards a learning-based PSF, and experimentation with various RL algorithms.

3.1.2 Model derivation

As presented in chapter 2, RL does not need to know the model of the environment a priori but can simply learn the dynamics and an optimal control strategy from trial and error. Due to high costs and safety concerns, it is not feasible to execute these trials on a physical FOWT. Thus a simulated environment is beneficial for training. Figure 3.1.1 shows that we need a model of the plant to be able to see how the system reacts to each action, which is the basis for the reward calculation. However, this model does not have to be analytical. It can also be represented by a neural network, or any other approach that represents the input-output dynamics of the system. On the other hand, our adapted PSF needs an analytical model, at least for the plant's unstable modes. The PSF presented here contains a plant approximation which is quite close to the simulation model, where only minor simplifications has been done to the plant. The simplification will be described in detail in section 3.2.

Accurate mathematical models for the dynamics of FOWTs are generally extremely complex and require a large amount of computing power to simulate. To be able to focus on the concept of combining RL with PSF, a simple approximation of a FOWT model was needed. While our model captures some of the real system dynamics, it is meant as a placeholder pending a more accurate model as a potential extension of the work. As shown in Figure 3.1.2, the structural model of the platform was approximated as a rigid body damped rod rotating about a fixed point. The platform was also equipped with a thruster below sea level to provide stabilization actuation, mainly aimed at suppressing platform oscillation.

A simplified engineering model for the wind and rotor dynamics was added to compute rotational torque on the rotor and the drag force acting on the platform. It was assumed that the force from wind acts as a point force at the top of the platform. This is a reasonable assumption because the platform itself is cylindrical and sufficiently aerodynamic. Wave forces and other disturbances are neglected.

The control objective then consists of optimizing power generation and rotor velocity while keeping the platform as vertical and stationary as possible — all of this using RL and PSF together.

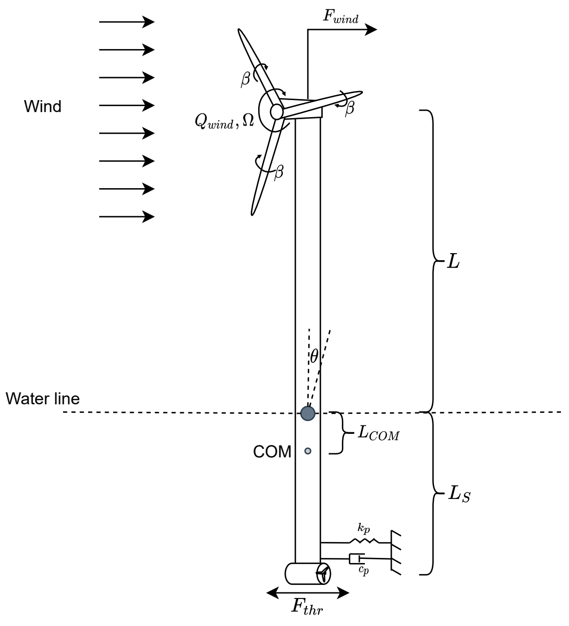


Figure 3.1.2: Simplified model of a floating wind turbine (illustration proportions are not to scale). An open-loop stable, damped platform with a thruster actuator in the bottom, and a rotor with a generator at the top.

Platform model

The derivation of the platform model is done using Lagrangian mechanics and is based on the illustration of our simplified wind turbine model in Figure 3.1.2. The kinetic energy is simply

$$T = \frac{1}{2} J_p \dot{\theta}^2 \tag{3.1.1}$$

where J_p is the inertia of the platform and $\dot{\theta}$ is the platform angular velocity. The potential energy of the platform is described by

$$V = \frac{1}{2} k_p (L_S \sin\theta)^2 + M g L_{COM} (1 - \cos\theta) \tag{3.1.2}$$

where θ is the platform angle, k_p is the spring constant of the platform motion, L_S is the distance from the platform rotation point (water line) to the platform thruster, L_{COM} is the distance from the platform rotation point to its center of mass, M is the platform mass including the rotor, and g is naturally the gravitational acceleration.

Using Lagrangian mechanics gives rise to the platform's equation of motion

$$\begin{aligned} J_p\ddot{\theta} + k_p L_S^2 \sin\theta\cos\theta + MgL_{COM}\sin\theta &= \underbrace{-c_p L_S \cos(\theta)\dot{\theta}}_{Damping force} + \underbrace{L_S F_{thr}}_{Thruster force} + L \underbrace{F_{wind}}_{Drag force} \\ \ddot{\theta} &= \frac{1}{J_p}\left(-k_p L_S^2 \sin\theta\cos\theta - MgL_{COM}\sin\theta - c_p L_S\cos(\theta)\dot{\theta} + L_S F_{thr} + LF_{wind}\right) \end{aligned} \tag{3.1.3}$$

where c_p is the damping constant of the platform motion, L is the distance from the platform's rotation point to the nacelle, F_{thr} is the force from the platform thruster, and F_{wind} is the drag force caused by wind.

Rotor Model

The angular acceleration of the rotor, $\dot{\Omega}$, depends on the torque generated by the wind, Q_{wind}, as well as the resistance torque from the generator, Q_{gen}. Using Equation 2.2.1 yields

$$\begin{aligned} J_r\dot{\Omega} &= Q_{wind} - Q_{gen} \\ \dot{\Omega} &= \frac{1}{J_r}\left(Q_{wind} - \frac{P_{gen}}{\Omega}\right) \end{aligned} \tag{3.1.4}$$

where J_r is the rotor inertia.

A problem with the generator model in Equation 2.2.1 occurs when the rotor velocity Ω becomes small, making the generator counter-torque Q_{gen} unreasonably large. The range of the values would lead to large derivatives, making the system numerically unstable, i.e. making our model invalid. This is both non-physical and undesirable. To avoid this numerical issue in the simulation, a limit is set on the amount of torque the generator can provide: $0 < Q_{gen} < 28.6$ MNm. The limit is set from intuition, as 15 MW is the maximum power generation and 5 rpm is the lowest rotational speed the turbine is rated for (Gaertner et al., 2020) ($\frac{15\text{ MW}}{5\text{ rpm}} \approx 28.6$ MNm). The limit is also reasonably close to the rated max generator torque of 20 MNm for the IEA 15MW. We choose to use control on the generator power P_{gen} rather than Q_{gen} because P_{gen} is easier to set the input range for (0 to 15 MW), and they are directly linked through Equation 2.2.1.

Equation 3.1.3 and Equation 3.1.4 still have two terms that are unaccounted for, F_{wind} and Q_{wind}. The model for this wind force and torque is adapted from the engineering model in Pedersen (2017, Model 4.6):

$$\begin{bmatrix} F_{wind} \\ Q_{wind} \end{bmatrix} = \begin{bmatrix} d_r|w| & -h(\Omega, w, u_p) \\ h(\Omega, w, u_p) & b_{dr}|\Omega| \end{bmatrix} \begin{bmatrix} w \\ -\Omega \end{bmatrix}$$

$$h(\Omega, w, u_p) = k_r(\cos(u_p)w - \sin(u_p)\Omega\ell) \tag{3.1.5}$$

where $u_p = \beta - \beta^*$ is the bias corrected blade pitch input, w is the wind speed and

$$k_r = \frac{2\rho A_p R}{3\lambda_*}, \quad b_{dr} = \frac{1}{2}\rho A(B^2\frac{16}{27} - C_P^*)(\frac{R}{\lambda_*})^3, \quad d_r = \frac{1}{2}\rho A C_F, \quad \ell = \frac{2R_p}{3}$$

where the constants are defined in Table 3.2.5. The point of optimal power extraction is realized at $u_p = 0$ through the use of MPPT control (Pedersen, 2017).

The validity of this simplified model was investigated by Pedersen (2017) for a 5MW turbine, which reports that "*given the simplicity of the model, superb performance is obtained*". We assumed that similar performance is obtained for the 15MW turbine. Pedersen (2017) emphasizes that care must be taken at low wind speeds as the theory behind the model might fail in such conditions. The definition of *low wind speeds* will naturally depend on the specifics of the turbine.

The wind speed w in Equation 3.1.5 has to be adjusted with regards to inflow and structural concerns as shown in Pedersen (2017, Figure 5.1). The engineering model was originally combined with a tip-loss corrected low frequency Dynamic Vortex Theory (DVT) model for the inflow model. A further simplification to this is assuming a *static* model for the inflow where it is simply assumed that $w_i \approx \frac{1}{3}w_0$ resulting in

$$w = w_0 - w_i - \dot{x} \approx \frac{2}{3}w_0 - L\cos(\theta)\dot{\theta} \tag{3.1.6}$$

where w_0 is the environment wind speed, w_i is the inflow, and w is the relative axial flux, i.e. the adjusted wind speed perceived by the rotor. We acknowledge that this is an extremely simplified approximation but it captures the main contributions to adjusted wind, which is the purpose.

Combined model

Combining Equation 3.1.3 for the platform and Equation 3.1.4 for the rotor gives the system:

$$\mathbf{x} = [x_1, x_2, x_3]^T = [\theta, \dot{\theta}, \Omega]^T$$
$$\mathbf{u} = [u_1, u_2, u_3]^T = [F_{thr}, u_p, P_{gen}]^T$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\theta} \\ \frac{1}{J_p}\left(-k_p L_S^2 \sin\theta\cos\theta - MgL_{COM}\sin\theta - c_p L_S \cos(\theta)\dot{\theta} + L_S F_{thr} + LF_{wind}\right) \\ \frac{1}{J_r}(Q_{wind} - \frac{P_{gen}}{\Omega}) \end{bmatrix}$$

$$= \begin{bmatrix} x_2 \\ \frac{1}{J_p}(-k_p L_S^2 \sin x_1 \cos x_1 - MgL_{COM}\sin x_1 - c_p L_S \cos(x_1)x_2 + L_S u_1 + LF_{wind}) \\ \frac{1}{J_r}(Q_{wind} - \frac{u_3}{x_3}) \end{bmatrix} \tag{3.1.7}$$

where F_{wind} and Q_{wind} are given by Equation 3.1.5.

A graphical overview of how the system is interconnected is shown in Figure 3.1.3. Note that this is graphic is a coarse overview and does not include all dependencies in the equations, e.g., the feedback between F_{wind} and $\dot{\theta}$.

To summarize the most relevant symbols: Q_{wind} is the rotational torque on the rotor from the wind. F_{wind} is drag force at the top of the turbine from the wind. Q_{gen} is the resistance torque from the generator. Ω is the angular velocity of the rotor. θ and $\dot{\theta}$ are the angle and angular velocity of the platform, respectively. Regarding control inputs, P_{gen} is the generated power, u_p is the bias-corrected pitch input, and F_{thr} is the platform thrust force.

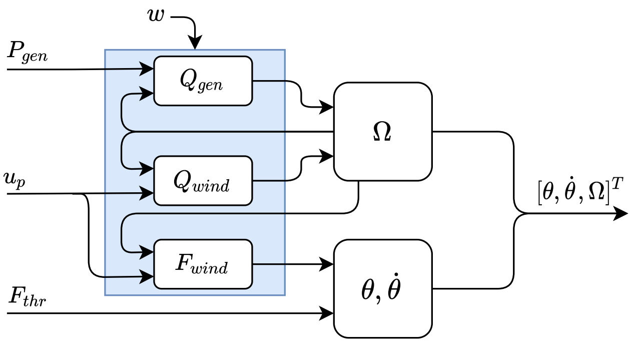


Figure 3.1.3: Block diagram overview of the equation system.

3.1.3 Predictive safety filter formulation

Building on section 2.4, we can now introduce our adapted nominal PSF formulation. Nominal in the sense that we do not consider disturbances, but note that our framework and methodology has the ability to extend to this realm as well. As a consequence, we are not able to prove stability in varying wind, which in reality is a big part of FOWT control. We therefore rely on the empirical evidence of the inherent robust stability of nominal MPC to handle slowly varying wind and other simplifications done to ensure computational feasibility.

Our approach differs in some areas from the original paper (Wabersich and Zeilinger, 2018*b*), with the main difference being that our PSF is not learning-based. The downside of this is of course the explicit expert knowledge needed to formulate state dynamics to obtain the discrete state trajectory $\Phi(\cdot)$. Our formulation is

$$\begin{aligned}
\min_{\mathbf{X},\mathbf{U}} \quad & (\mathbf{u}_0 - \mathbf{u}_{\mathcal{L}})^\intercal \mathbf{R} (\mathbf{u}_0 - \mathbf{u}_{\mathcal{L}}) && \text{(3.1.8a)}\\
s.t. \quad & \\
\mathbf{x}_{k=0} &= \mathbf{x}_t && \text{(3.1.8b)}\\
\mathbf{x}_{k+1} &= \Phi(\tau_{t,k}, \mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k) \qquad \forall k \in \mathbb{I}_{\leq N} && \text{(3.1.8c)}\\
\mathbf{H}_x \mathbf{x}_{k+1} &\leq \mathbf{h}_x && \text{(3.1.8d)}\\
\mathbf{H}_u \mathbf{u}_k &\leq \mathbf{h}_u && \text{(3.1.8e)}\\
\mathbf{x}_{N+1} &\in \mathcal{T} \subseteq \mathcal{S} && \text{(3.1.8f)}
\end{aligned}$$

where $\mathbf{R}$ is the quadratic cost matrix, $\mathbf{u}_{\mathcal{L}}$ is the RL proposed input, and the rest of the variables are defined as in Equation 2.4.19. The main difference from the non-linear MPC can be seen in the cost function, which is only dependent on the first input in the control sequence $\mathbf{u}_0$ and the proposed control input $\mathbf{u}_{\mathcal{L}}$. In addition we have limited the constraints from Equation 2.4.19d to polytopic constraints as described in subsection 2.1.1. At each time step, the PSF finds a backup trajectory which minimizes the distance between $\mathbf{u}_0$ and $\mathbf{u}_{\mathcal{L}}$. Deviations between the two will occur when the input would lead to constraint violations or the state system trajectory does not enter the terminal set at the end of the horizon. If the RL's proposed input is to be allowed directly, then $\mathbf{u}_0 - \mathbf{u}_{\mathcal{L}}$ should be zero. In this scenario, the value of $\mathbf{R}$ does not matter. However, once filtering is needed, the shape of the cost function determines $\mathbf{u}_0$. A natural approach is to let $\mathbf{R}$ normalize the elements of $\mathbf{u}_0$.

Conceptually we are now close to the quadratic program-based CBF proposed in Ames et al. (2016). One could argue that the presented PSF formulation is a special case of CFB, the difference being that the safe set is implicitly defined. The PSF approximates the safe set $\tilde{\mathcal{S}} \subseteq \mathcal{S}$, where the temporal span of the horizon is therefore related to how conservative the PSF is. If we let the horizon tend toward infinity, it will span the entire safe set. At this point, the PSF would be at its least conservative. A naive implementation with an infinite horizon is, of course, computationally intractable which is why we want the largest possible terminal set. If we only have a single step in the PSF, the optimization is a direct projection onto the terminal set. Leaving us with adjusting the length of N as a trade-off between conservativeness and computational complexity. Figure 3.1.4 is an illustration of the aforementioned sets.

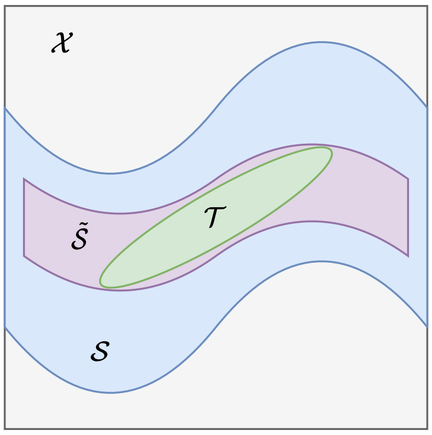


Figure 3.1.4: Illustrates the conceptual relation between the different sets related to the PSF. The grey area is the system constraints $\mathcal{X}$, the blue is the true safe set $\mathcal{S}$, the purple is the approximated safe set $\tilde{\mathcal{S}}$, and the green is the terminal set $\mathcal{T}$. Extending the temporal span of the PSF horizon can increase the size of $\tilde{\mathcal{S}}$.

One benefit of using the PSF in combination with an RL agent, instead of a traditional MPC, is that the tasks of optimality and safety are separated. The PSF is left with the simple task of keeping the state within the safe set, while the task of optimality is left to the RL agent. Due to its modularity and separation the optimizing controller could take any form. Recursive feasibility proof outlined in subsubsection 2.4.2 could be reapplied here in the same manner. This property is ensured if the initial problem is feasible and and the plant approximation in PSF is sufficiently accurate.

3.1.4 Obtaining the terminal set

Working our way from the system dynamics there are a few problems which come in the way of obtaining the terminal set. To be able to apply the theory from subsection 2.4.1, we must transform the nonlinear system into a linear one. This is done through employing the robust control methods discussed, also in the same section. We also need to handle a few other details to be able to construct the ellipsoidal terminal set, this is discussed in the following parts.

Constructing the linear uncertainty system

As seen in Equation 2.1.3 any ODE can be approximated linearly around a point v_0. Using the combination of linearization and lifting the affine system as displayed in Equation 2.1.5, we can reformulate the ODE as

$$\begin{aligned}\dot{\mathbf{x}} &= \nabla^\mathsf{T}\mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}}(\mathbf{x}-\mathbf{x}_{lin}) + \nabla^\mathsf{T}\mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}}(\mathbf{u}-\mathbf{u}_{lin}) + \mathbf{f}|_{\mathbf{z_{lin}}} \\ &= \nabla^\mathsf{T}\mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}}\mathbf{x} + \nabla^\mathsf{T}\mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}}\mathbf{u} + \underbrace{\mathbf{f}|_{\mathbf{z}_{lin}} - \nabla^\mathsf{T}\mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}}\mathbf{x}_{lin} - \nabla^\mathsf{T}\mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}}\mathbf{u_{lin}}}_{\mathbf{b}} \end{aligned} \tag{3.1.9}$$

$$\dot{\tilde{\mathbf{x}}} = \begin{bmatrix} \nabla^\mathsf{T}\mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}} & \mathbf{f}|_{\mathbf{z}_{lin}} - \nabla^\mathsf{T}\mathbf{f}_\mathbf{x}|_{\mathbf{z}_{lin}}\mathbf{x}_{lin} - \nabla^\mathsf{T}\mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}}\mathbf{u}_{lin} \\ \mathbf{0} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} + \begin{bmatrix} \nabla^\mathsf{T}\mathbf{f}_\mathbf{u}|_{\mathbf{z}_{lin}} \\ \mathbf{0} \end{bmatrix} \mathbf{u} \tag{3.1.10}$$

$$\dot{\tilde{\mathbf{x}}} = \mathbf{A}(\boldsymbol{\delta})\tilde{\mathbf{x}} + \mathbf{B}(\boldsymbol{\delta})\mathbf{u} \tag{3.1.11}$$

We can now use the robust control principles from subsection 2.4.1 to handle the nonlinearities if we can express the system matrices as affine functions of the uncertainty $\boldsymbol{\delta}$. The number of LMIs introduced to the formulation in Equation 2.4.1 is 2^p, where p is the dimension of $\boldsymbol{\delta}$. Due to the exponential growth in system "extremes" having to be stabilized, we want to find a polytope with as few vertices as possible which still spans the matrix space. However, only reducing the dimension of $\boldsymbol{\delta}$, could result in an overly coarse approximation and reducing the chance find a feedback controller satisfying the LMIs. Therefore, we also want to tighten the constraints around the function. The problem is closely related to the minimal enclosing box, or the more general minimal enclosing polytope (Panigrahy, 2004). Further expansion in this direction is possible but not considered in this thesis.

When the system dynamics are of the form of Equation 3.1.11, an orthotope (hyperrectangle) $\boldsymbol{\Delta}$ can be created through optimization. Re-expressing the system matrices in terms of $\delta_{i,j}(\mathbf{z}) = \delta_{i,j}$ gives

$$\begin{bmatrix}\mathbf{A}(\boldsymbol{\delta}) & \mathbf{B}(\boldsymbol{\delta})\end{bmatrix} = \begin{bmatrix} \delta_{1,1} & \delta_{1,2} & \dots & \delta_{1,n_x+n_u+1} \\ \delta_{2,1} & \delta_{2,2} & \dots & \delta_{2,n_x+n_u+1} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n_x,1} & \delta_{n_x,2} & \dots & \delta_{n_x,n_x+n_u+1} \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{3.1.12}$$

The range of the matrices elements $\delta_{i,j}$ is found through the optimization

$$\begin{aligned} \left[\underline{\delta_{i,j}}, \overline{\delta_{i,j}}\right] &= \left[\min_{\mathbf{z}} \quad \delta_{i,j}, \max_{\mathbf{v}} \quad \delta_{i,j}\right], i \in \mathbb{I}_{\leq n_x}, j \in \mathbb{I}_{\leq n_x+n_u+1} \\ & s.t. \\ \mathbf{H}_z\mathbf{z} &\leq \mathbf{h}_z \end{aligned} \tag{3.1.13}$$

where the constraint $\mathbf{H}_z\mathbf{z} \leq \mathbf{h}_z$ does not need to coincide with the PSF constraints. The procedure creates the box $\boldsymbol{\Delta} = \in \mathbb{R}^p$, where $p = n_x(n_x+n_u+1)$. This approach scales rater poorly, but this is under the assumption that every entry varies with $\mathbf{z}$. However, this is often not the case. To increase the probability of finding a linear feedback controller and the correlated terminal set, we could adjust the range of the external parameters. A smaller range would lead to a more specific solution with a large ellipsoidal set.

In a sense, we have linearized the system around every possible state and stabilized the entire set at the same time. As we do not consider our system states' derivatives, we cannot apply saturation limits to our terminal set in practice. The extension is trivial in theory but will further decrease the size of the terminal set. The resulting system uncertainty is now an outer approximation of the nonlinear system, leading to a conservative controller (Bemporad and Morari, 1999). However, this now lends itself to robust linear feedback control, thus realizing our terminal set. Note that we merely require that such a system theoretically exists, as the feedback gain is not explicitly used. As a result, we may use both the discrete or the continuous-time implementation. In our implementation, we used the latter to avoid the challenges of choosing a discretization time-step.

Details concerning the robust ellipsoid

The formulation in Equation 2.4.13 is centered at the origin, but the rotors rotation Ω is bound above 0. We therefore need to move the ellipsoid inside the constraints. Furthermore, the equation in Equation 2.4.15 assumes that the constraints are equidistant from the origin. More precisely, it constrains the ellipsoid symmetrically around the origin. The closest constraining hyperplane is in essence mirrored around the ellipsoid center. The method used here tries to handle both of these concerns by finding a center $(\mathbf{x}_{c_0}, \mathbf{u}_{c_0})$ where the system is at steady state and maximizes the distance to each constrain hyperplane. The latter coincides with minimizing the distance given that cost is quadratic, leaving us with following optimization problem

$$\begin{aligned}
\mathbf{x}_{c_0}, \mathbf{u}_{c_0} = &\arg\min_{\mathbf{x},\mathbf{u}} && \begin{aligned}&(\mathbf{H}_x\mathbf{x} - \mathbf{h}_x)^\intercal \mathbf{Q}(\mathbf{H}_x\mathbf{x} - \mathbf{h}_x) + \\ &(\mathbf{H}_u\mathbf{u} - \mathbf{h}_u)^\intercal \mathbf{R}(\mathbf{H}_u\mathbf{u} - \mathbf{h}_u)\end{aligned} && \text{(3.1.14a)} \\
&s.t. && && \\
&\dot{\mathbf{x}} = 0 && && \text{(3.1.14b)} \\
&\mathbf{H}_x\mathbf{x} \le \mathbf{h}_x && && \text{(3.1.14c)} \\
&\mathbf{H}_u\mathbf{u} \le \mathbf{h}_u && && \text{(3.1.14d)}
\end{aligned}$$

where $\mathbf{R}$ and $\mathbf{Q}$ are normalizing diagonal matrices. The diagonal elements $\mathbf{r}$ of $\mathbf{R}$ are found with the optimization problem

$$\mathbf{r} = \max \mathbf{u} - \min \mathbf{u} \quad s.t. \quad \mathbf{H}_u\mathbf{u} \le \mathbf{h}_u \tag{3.1.15}$$

The elements of $\mathbf{Q}$ are found in a similar fashion. Obtaining our new origin, we now move the system according to Equation 2.1.11 and Equation 2.1.6.

3.2 Set-up

This section contains the specifics of our setup and more detailed information on how the work was conducted. It also addresses some of the weaknesses of the model and justification for choices made during the work.

3.2.1 Frameworks, packages, and hardware

The RL framework of the software was developed using the OpenAI Gym toolkit, and Stable Baselines3 was used for RL algorithm implementations and high-level control of agent training.

A large focus was placed on using open-source frameworks and packages. While many programming languages offer some deep learning toolkits, the ML community is mostly based around Python. This is in contrast to control engineering tasks, where MATLAB is often the preferred tool. As an example, the most popular framework for LMIs is YALMIP (Löfberg, 2004), which is implemented in MATLAB. As Python and MATLAB are able to communicate, a dual language implementation is possible, and a working prototype was implemented using this. However, a native python implementation is clearly beneficial with respect to licenses, execution times, and ease of installation. Through the use of the CVXPY framework (Diamond and Boyd, 2016; Agrawal et al., 2018), we were able to eliminate the MATLAB dependencies. CVXPY is a Python-embedded modeling language for convex optimization problems that offers support for the most common solvers. While open-source solvers are available (e.g., CVX OPT (Vandenberghe, 2010)), the commercial solver MOSEK ApS (2019) was used to solve the terminal set problem in our implementation. Other solvers should be capable, however MOSEK converged on solutions significantly faster and with a higher success rate, leaving it as the obvious choice. MOSEK is also available through a free academic license, so licensing was not considered an issue.

The PSF was implemented in Casadi (Andersson et al., 2019), an open-source tool for nonlinear optimization. While implemented in C, it communicates efficiently with our framework through a Python interface. The core of Casadi is its symbolic framework, which allows automatic differentiation. This can be used to efficiently construct gradients, which can again be parsed to a wide variety of solvers. To solve the Non-Linear Program (NLP), the interior point solver IPOPT (Biegler and Zavala, 2009) was chosen due to its versatility.

As previously explained, our model was based on the IEA 15MW RWT. To investigate the validity of our simplified model and find appropriate coefficients for our equations, system identification had to be performed. This was done by comparing our simple model to OpenFAST simulations of the RWT. OpenFAST (Jonkman et al., 2021) is an open-source, multi-physics, multi-fidelity tool for simulating the coupled dynamic response of wind turbines and includes a high fidelity model of the IEA 15MW turbine.

The fourth order Runge-Kutta-Fehlberg method (Fehlberg, 1970) was used as the numerical solver to simulate the dynamics of the turbine. The Butcher table for this method is shown in Table 3.2.1. Using a more complex RK method than the simple forward Euler method increases the stability margin of the simulation for larger time-steps, allowing faster simulations. A step size of 0.1 was used.

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$

Table 3.2.1: Butcher tableau for Runge-Kutta-Fehlberg method 4(5).

Two different hardware setups were used during the work. The RL agents without PSF were trained on a Dell OptiPlex 7060 computer with an Intel Core i7-8700 CPU and 32 GB RAM. Introducing the PSF increased the computation time significantly, and in order to reduce the time needed for training, processes were deployed on the IDUN High Performance Computing Group (Själander et al., 2019). Although this did not speed up the training for any single agent, it allowed for further parallelization and sped up the acquisition of results.

3.2.2 Floating offshore wind turbine model

Our model is meant to be a coarse approximation of the 15MW FOWT presented by IEA (Allen et al., 2020) capturing the general behavior of the rotor and platform. Control inputs for the system were chosen to be $[F_{thr}, u_p, P_{gen}]^T$. In the *plant* block in Figure 3.1.1, i.e., the model used by the RL agent, the inputs are low-pass filtered to create more realistic behaviors for actuators. The *plant approximation* block in the PSF implements an approximation of this behavior through rate saturations. The time constants in the low-pass filters and the saturation limits for each input can be found in Table 3.2.5 and 3.2.9.

The turbine has a setpoint curve for both generated power and rotor velocity based on the wind speed. These are shown in Figure 3.2.1 and Figure 3.2.2. It is assumed that these curves are based on the wind speed w_0 and not the adjusted wind speed w.

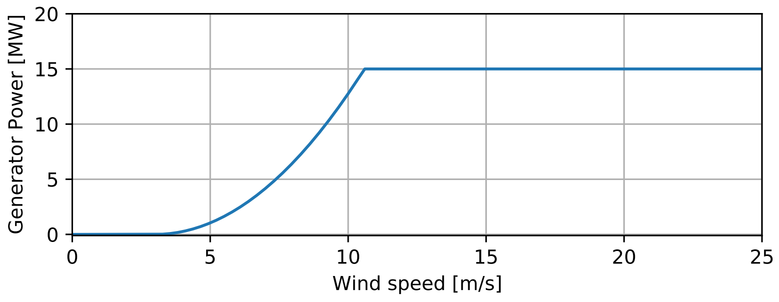


Figure 3.2.1: Generator power setpoint curve (Gaertner et al., 2020).

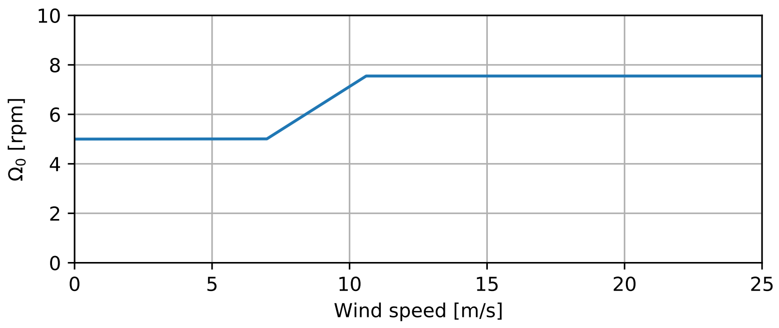


Figure 3.2.2: Ω setpoint curve (Gaertner et al., 2020).

The constraints for the system are set to be as shown in Table 3.2.2. The constraints on Ω and P_{gen} are set based on the RWT specifications. When choosing the constraints for F_{thr}, bollard pull was used as a basis. Ajay Menon (2021) reports that general values of bollard pull for medium-sized tugs used in port operations can range between 500 to 600 kN. We acknowledge that these numbers are intended for tug boats and not our type of actuation, but it was assumed that we could equip the platform with a max thrust force of 500 kN. The constraints on u_p and θ are simply set based on a combination of reason, feasibility, and data from Gaertner et al. (2020).

Table 3.2.2: System constraints.

Variable	Lower Constraint	Upper Constraint	Unit
u_p	-4	20	[deg]
F_{thr}	$-5 \cdot 10^5$	$5 \cdot 10^5$	[N]
P_{gen}	0	$15 \cdot 10^6$	[W]
Ω	5	7.6	[rpm]
θ	-10	10	[deg]

System identification for platform

For our model to have the correct behavior of the turbine, we had to find the coefficients for Equation 3.1.3. This was done by rewriting the equation as

$$\ddot{\theta} = C_1 \sin\theta\cos\theta + C_2\sin\theta + C_3\cos(\theta)\dot{\theta} + C_4 F_{thr} + C_5 F_{wind} \tag{3.2.1}$$

The response of the IEA 15MW FOWT (Allen et al., 2020) was then analyzed in OpenFAST and the coefficients were fitted to this data using 1-norm regression. With this, we can construct a regression problem, viz.

$$\min|\ddot{\theta} - \mathbf{C}^{\intercal}\mathbf{y}|, \qquad \mathbf{C} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_5 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} \sin\theta\cos\theta \\ \sin\theta \\ \cos(\theta)\dot{\theta} \\ F_{wind} \end{bmatrix} \tag{3.2.2}$$

The steady state for the platform angle of the RWT model is approximately $\theta_{offset} = -1.5\,\mathrm{deg}$ (leaning into the wind), while our platform is centered at $\theta = 0$. The offset was subtracted from the platform angle $\theta = \theta_{RWT} - \theta_{offset}$ to counteract its influence. The reason for using 1-norm as loss is due to its ability to suppress outliers (Zhang and Luo, 2015). Some of the OpenFAST variables are coarsely differentiated numerically, leaving it necessary to also run the simulation multiple times for further noise suppression. The data was gathered from 25 runs with different initial angles. Due to large transients when initializing the turbine with no movements, the turbine was given a predetermined rotation and wind speed of 7.55 rpm and 12 m/s, respectively. The wind speed was then steadily changed to random wind speed between 10 and 25 with a ramp function to further excite the system. The coefficients $C_{1\dots5}$ in Equation 3.2.1 were then fine-tuned manually to fit the step response of the platform from OpenFAST with regards to the period and damping of the θ-oscillations. They can be found in Table 3.2.3.

Table 3.2.3: Platform coefficients.

Coefficient	Value
C_1	4.45
C_2	-4.49
C_3	$-5.55 \cdot 10^{-3}$
C_4	$C_5 \cdot \frac{L_{thr}}{L}$
C_5	$9.92 \cdot 10^{-10}$

OpenFAST offers a wide range of data output tags. The relevant ones and their relation, name and meaning can be seen in Table 3.2.4.

Table 3.2.4: Data tags in OpenFAST used to obtain the platform and thrust coefficients.

Model	Tag Name	Description
θ	PtfmRDyi	Platform pitch tilt angular (rotational) displacement
$\dot{\theta}$	PtfmRVyi	Platform pitch tilt angular (rotational) velocity
$\ddot{\theta}$	PtfmRAyi	Platform pitch tilt angular (rotational) acceleration
F_{wind}	RotThrust	Low-speed shaft thrust force

The models step response

The step response of the platform without wind can be seen in Figure 3.2.3. The proposed model is relatively static in the frequency of θ. However, the OpenFAST simulation shows that the reference model reacts with different frequencies to different initializations in θ. Notice that the proposed model has a frequency somewhere in between, lagging behind when the pitch angle is small but ahead at large angles. In both cases, the model is erroneous but is chosen as an intermediate estimation of the RWT.

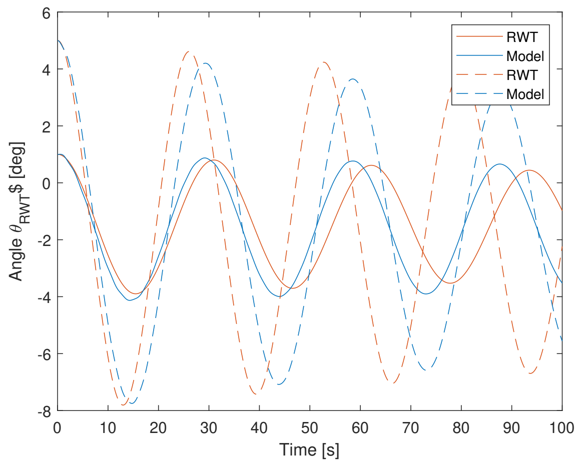


Figure 3.2.3: Response of the proposed model (Model) and the reference model (RWT) with two different initialization. The proposed model acts intermediate estimation of the RWT simulation when no wind is present. Steady state offset is taken into account.

With a change in F_{wind}, we can further investigate the response of our system under the influence of wind. The change in axial force is induced by creating a step from no wind to 12 m/s, which is then measured as a force in the nacelle. Note that the step is created in *wind speed* and not in F_{wind}, so the axial force will take a different shape. The responses can be seen in Figure 3.2.4. We see that our model's initial response is comparable, but it fails to capture the wind's dampening effect, resulting in a quite different response. As a consequence, our model oscillates heavily under the influence of a wind force. Nevertheless, our model oscillates around the same point, which goes to show that our platform model attains some of the basic properties of the system, like the steady-state effect of wind.

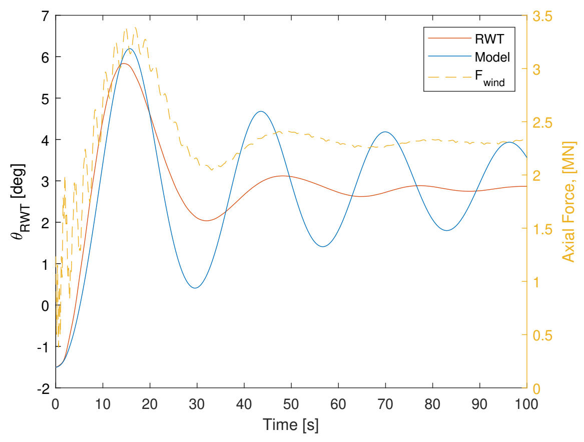


Figure 3.2.4: Response to changes in wind from 0 m/s to 12 m/s. The step is instantaneous in wind, but not in the axial force. The axial force applied at nacelle of the proposed model and the references model(RWT) simulated OpenFAST. Steady state offset is taken into account.

3.2.3 Reinforcement learning

Our control problem has a continuous state and action space, much like in Meyer (2020). Teigen (2020) shows that the PPO algorithm is preferable for such applications, which is why we use this algorithm in our implementation. We formulate the problem as an episodic event where the wind conditions are varied within certain ranges between episodes. The agent needs to train for multiple episodes to be able to learn the dynamics. This requires us to define how an episode is started and when it is done.

An episode is started with a random initial wind speed within a given range. A choice was made to start the turbine in a steady state given this wind speed. Starting the turbine in an initial state of zero rotor velocity and platform angle would lead to a large transient on start-up. Instantly applying a step in the wind from 0 to e.g., 20 m/s on a stationary turbine is neither in the domain of the control problem in this thesis nor within the limits of our model's validity. Control of the turbine outside its rated wind speeds of 5-25 m/s is also outside the scope of this thesis. This leads us to a solution where the desired wind is applied, and the turbine is initialized in a steady-state solution within the constraints in Table 3.2.2. The steady state is calculated with the assumption of F_{thr} being set to zero. Note that the steady state is not necessarily the optimal state given the conditions. The only requirement is that it is within the constraints of the system.

This gives a realistic starting point for the RL agent in each episode. The task for the RL agent will then be to adjust the propeller thrust, blade pitch angle, and generator power generation to bring it to an optimal state and continue stabilization with varying wind.

Similarly, the end of each episode needs to be defined. An episode is said to be

Table 3.2.5: Model parameters and variables.

Symbol	Description	Value	Unit
β	Rotor blade pitch angle		[rad]
β^*	MPPT Optimal blade pitch angle		[rad]
u_p	Bias corrected pitch input	$\beta - \beta^*$	[rad]
F_{thr}	Force from platform thruster		[N]
P_{gen}	Generated power		[W]
w_0	Environment wind speed		$[\frac{\text{m}}{\text{s}}]$
w_i	Inflow wind	$\frac{1}{3}w_0$	$[\frac{\text{m}}{\text{s}}]$
w	Wind speed (Relative axial flux)	$w_0 - w_i - \dot{x}$	$[\frac{\text{m}}{\text{s}}]$
θ	Platform angle		[rad]
Ω	Rotor angular velocity		$[\frac{\text{rad}}{\text{s}}]$
Ω_0	Setpoint for rotor angular velocity	0.75	$[\frac{\text{rad}}{\text{s}}]$
$F_{thr,max}$	Max force from platform thruster	500 000	[N]
τ_{thr}	Platform thruster time constant	2	[s]
$u_{p,max}$	Max bias corrected pitch input	0.3491 (20 deg)	[rad]
$u_{p,min}$	Min bias corrected pitch input	$-0.2u_{p,max}$ (-4 deg)	[rad]
τ_u	Blade pitch time constant	1.3	[s]
$P_{gen,max}$	Maximum power generation	15 000 000	[W]
τ_{gen}	Generator power time constant	2	[s]
B	Tip loss parameter	0.97	-
R	Rotor radius	120	[m]
A	Rotor area	πR^2	[m^2]
R_p	Tip loss corrected rotor radius	$B \cdot R$	[m]
A_p	Tip loss corrected rotor area	πR_p^2	[m^2]
ρ	Air Density	1.225	$[\frac{\text{kg}}{\text{m}^3}]$
C_P^*	Power coefficient	0.489	-
C_F	Force coefficient	0.8	-
λ_*	Tip speed ratio	9	-
J_r	Rotor inertia	4.06890357e+07	[kg m^2]
L	Distance from water line to nacelle	144.45	[m]
L_{thr}	Distance from water line to platform thruster	50	[m]

done if either of the following conditions is met.

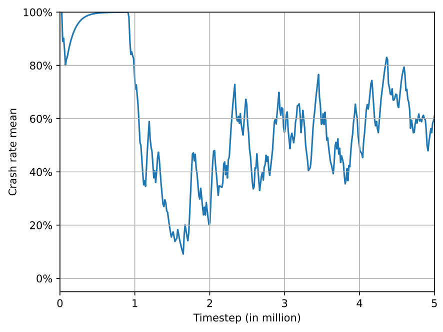
- A maximum number of timesteps is executed (3000 steps).
- The turbine has crashed, meaning

  The platform angle, θ, is outside the $[-10, 10]$ degrees range, or

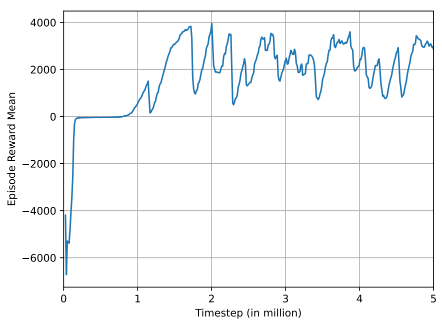
  The angular velocity of the rotor, Ω, is above 10 RPM, or

  The angular velocity of the rotor, Ω, is below 3 RPM.
- The PSF fails to solve the optimization problem.

The reason for the crash conditions on Ω being different from the constraints in Table 3.2.2 is that we want to allow some slack for the PSF. This is done because the simplifications in the PSF might cause it to allow the system to go slightly outside the constraints. The wider constraints were also observed to be beneficial for the RL agent as it made the optimization problem easier to solve. It is also natural to widen the constraints for the RL agent to facilitate learning by allowing more exploration, especially in the case without PSF. Figure 3.2.5 shows the training progress with the tighter constraints, and we clearly see that the agent does not converge to an optimum with respect to the reward. Neither is it able to learn that it is beneficial to keep the system within the constraints.



(a) Episode crash rate mean (smoothed).



(b) Episode reward mean.

Figure 3.2.5: Training progress for RL agent trained with episode conditions same as constraints in PSF (Table 3.2.2).

We stress that throughout this report we refer to both crash prevention, which is defined by the conditions above, and constraint satisfaction, which is defined as keeping the system within the tighter constraints in Table 3.2.2. By defining an episode as crashed differently than the violating the PSF constraints, we reduce our proof of constraint satisfaction to crash prevention.

Environment setup

Figure 3.2.6 shows the structure of the environment implementation in software, where everything is based around a base environment containing a turbine model as well as a PSF. This is then inherited by sub-environments that add various wind models, one for varying wind and one where the wind speed is constant. The sub-environments of the variable-wind environments enable the setting of wind mean and amplitude ranges. This is done to make it easy to extend the framework with different and more challenging wind models.

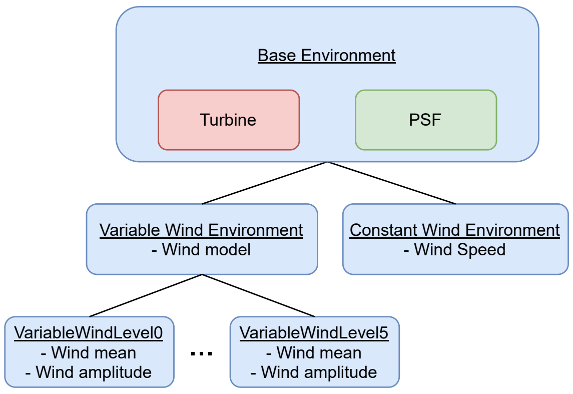


Figure 3.2.6: Overview of the RL environment framework.

Reward function

As explained in section 2.3, the RL agent optimizes its policy based on a reward. Therefore, shaping of a reward function is an paramount to the agent's performance and we define it as

$$r^{(t)} = r_{\theta}^{(t)} + r_{\dot{\theta}}^{(t)} + r_{\Omega}^{(t)} + r_{\dot{\Omega}}^{(t)} + r_{P}^{(t)} + r_{PSF}^{(t)} \tag{3.2.3}$$

Each term in the reward is presented below. See Figure 3.2.9 for plots of the first five terms' contribution to the reward function. The coefficients used within each of the terms are listed in Table 3.2.6. Note that the units of the variables in the reward function are changed from standard SI units to make the curve shaping more intuitive during development.

The main goal of the agent is to optimize the generated power, thus we introduce the following term

$$r_{P}^{(t)} = e^{-\gamma_P |P - P_0|} - \gamma_P |P - P_0| \tag{3.2.4}$$

where P and P_0 are given in MW. The curve for P_0 is can be seen in Figure 3.2.1.

$r_P^{(t)}$ is needed to guide the agent towards keeping P_{gen} as close to the setpoint curve as possible. The curve is chosen as a combination of a decaying exponential and an absolute value term. The decaying exponential is chosen to introduce the initial rapid decay. The absolute value term is used to avoid a derivative of zero far from the desired value as this might make it hard for the agent to optimize the reward function by gradient ascent.

The turbine only operates when keeping Ω within a certain range. There is also a desired setpoint curve for Ω. We want to keep Ω as close to this curve as possible while still optimizing power generation and add the term

$$r_{\Omega}^{(t)} = e^{-\gamma_\Omega |\Omega - \Omega_0|} - \gamma_\Omega |\Omega - \Omega_0| \tag{3.2.5}$$

where Ω is given in RPM. The curve for Ω_0 can be seen in Figure 3.2.2. The shape of $r_{\Omega}^{(t)}$ is chosen to be the same as for $r_{P}^{(t)}$ because the functions serve the same purpose, but for different variables.

In addition to keeping Ω close to the setpoint, we want to avoid rapid oscillations in Ω. This can be realized through adding a penalty term for $\dot{\Omega}$,

$$r_{\dot{\Omega}}^{(t)} = -\gamma_{\dot{\Omega}}\dot{\Omega}^2 \tag{3.2.6}$$

where $\dot{\Omega}$ is given in RPM per second. $r_{\dot{\Omega}}^{(t)}$ also indirectly reduces rapid oscillations in u_p. These rapid oscillations are undesirable because they increase wear and tear on the turbine, especially the actuators.

Our control problem also consists of keeping the platform as vertical as possible while keeping it stationary and minimizing oscillations. Hence, terms for θ and $\dot{\theta}$ were also added.

$$r_{\theta}^{(t)} = e^{-\gamma_\theta|\theta|} - \gamma_\theta|\theta| \tag{3.2.7}$$

$$r_{\dot{\theta}}^{(t)} = -\gamma_{\dot{\theta}}\dot{\theta}^2 \tag{3.2.8}$$

where θ is given in degrees and $\dot{\theta}$ is given in degrees per second.

For the pure RL part of the problem, this would be all the terms needed in the reward function but when adding the PSF we also need to consider the corrected actions. To prevent the agent from only relying on the PSF, we add a negative reward for actions that would violate the constraints of the system given by

$$\begin{aligned} r_{PSF}^{(t)} &= -\gamma_{PSF}\left\|\frac{\mathbf{u}_{\mathcal{L}} - \mathbf{u}_0}{\mathbf{u}_{max}}\right\|_1 \\ &= -\gamma_{PSF}\left(\left|\frac{F_{thr} - F_{thr,PSF}}{F_{thr,max}}\right| + \left|\frac{u_p - u_{p,PSF}}{u_{p,max}}\right| + \left|\frac{P_{gen} - P_{gen,PSF}}{P_{gen,max}}\right|\right) \end{aligned} \tag{3.2.9}$$

where the subscript *PSF* symbolizes the corrected action given by the PSF.

$r_{PSF}^{(t)}$ is of course, only applied if the PSF is used during training. In the case of training an agent without the PSF, $r_{PSF}^{(t)}$ is always set to zero. The magnitude of $r_{PSF}^{(t)}$, can be relatively large because the difference between the PSF-corrected action and agent action will be zero as long as the action keeps the system within the constraints. Thus γ_{PSF} can be relatively large. The intention is for $r_{PSF}^{(t)}$ to dominate the reward initially and teach the agent to make *safe* actions, then allow the agent to optimize further through the other terms when it has learned safe behavior and $r_{PSF}^{(t)}$ is small.

As seen in Figure 3.2.9, most of the terms in the reward function will approximately be in the interval [-1,1] for reasonable values of each variable. Equation 3.2.3 gives a maximum theoretical value for the reward of 3 per timestep, or 9000 per episode.

Table 3.2.6: Coefficients in reward function.

Symbol	Definition	Value
γ_θ	Coefficient for θ reward	0.12
$\gamma_{\dot{\theta}}$	Coefficient for $\dot{\theta}$ reward	3
γ_Ω	Coefficient for Ω reward	0.285
$\gamma_{\dot{\Omega}}$	Coefficient for $\dot{\Omega}$ reward	4
γ_P	Coefficient for power reward	0.1
γ_{PSF}	Coefficient for PSF reward	5

The following paragraphs provide justification for each of the terms in Equation 3.2.3. Several different reward functions were explored, all of them a simple sum of terms representing desired behavior. In many of the versions a survival term $r^{(t)}_{survival}$ or a crash term $r^{(t)}_{crash}$ was added to encourage the agent to avoid crashing. These are defined as

$$\begin{aligned} r^{(t)}_{survival} &= 1 \qquad\qquad \forall\, t \\ r^{(t)}_{crash} &= \begin{cases} \text{-1000} & \text{if turbine has crashed} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The training progress for each of these functions can be seen in Figure 3.2.7. Note that the theoretical maximum values for the rewards are different for each $R_{1\ldots7}$, so the comparison has to be made on qualitative progress and not exact converged value.

The proposed reward functions are

- R_1: $r^{(t)}_P$
- R_2: $r^{(t)}_P + r^{(t)}_{crash}$
- R_3: $r^{(t)}_{\dot{\theta}} + r^{(t)}_\Omega + r^{(t)}_P + r^{(t)}_{PSF} + r^{(t)}_{survival}$
- R_4: $r^{(t)}_\theta + r^{(t)}_{\dot{\theta}} + r^{(t)}_\Omega + r^{(t)}_P + r^{(t)}_{PSF} + r^{(t)}_{survival}$
- R_5: $r^{(t)}_\theta + r^{(t)}_{\dot{\theta}} + r^{(t)}_\Omega + r^{(t)}_P + r^{(t)}_{PSF} + r^{(t)}_{crash} + r^{(t)}_{survival}$
- R_6: $r^{(t)}_\theta + r^{(t)}_{\dot{\theta}} + r^{(t)}_\Omega + r^{(t)}_{\dot{\Omega}} + r^{(t)}_P + r^{(t)}_{PSF} + r^{(t)}_{survival}$
- R_7: $r^{(t)}_\theta + r^{(t)}_{\dot{\theta}} + r^{(t)}_\Omega + r^{(t)}_{\dot{\Omega}} + r^{(t)}_P + r^{(t)}_{PSF}$

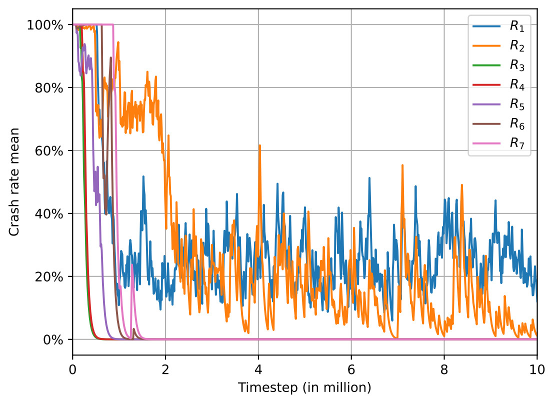
R_1 and R_2 were explored as attempts to keep the reward function as simple as possible, only centering the attention on the main objective of a turbine, optimizing power generation. As Figure 3.2.7a shows, this was not successful, and the agent was not able to learn not to crash.

R_3 adds terms for some of the other control objectives, namely making Ω follow the setpoint curve and keeping the platform as stationary as possible though $\dot{\theta}$. A survival term was also added to encourage the agent not to crash. We will come back to the PSF term later. A problem with the response of the agent trained using R_3 was that it used the platform thruster the opposite way of intended, i.e., increasing θ. A term for θ was therefore added in R_4. Comparing the training progress of agents with R_5 to R_4 in Figure 3.2.7, we see that adding a crash reward was not beneficial and actually slowed down learning.

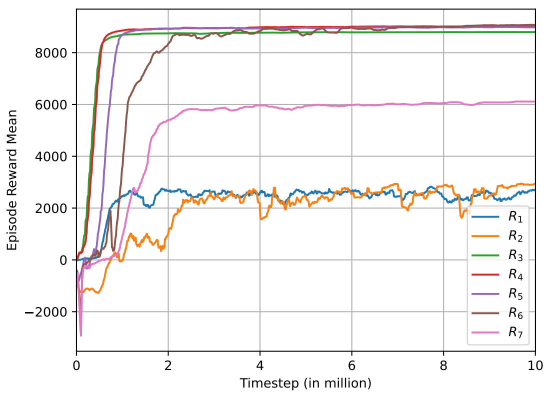
R_3 and R_4 both had similar progress, appearing to be the ones giving the fastest learning. One problem with these was that they resulted in agents using oscillatory inputs, thus oscillatory response in Ω. A term for $\dot{\Omega}$ was added in R_6 to counteract this unwanted behavior. This addition slowed down learning, as seen for R_6, but resulted in a better qualitative response in u_p and Ω reducing wear and tear on the turbine. The improvement in response for Ω after adding a term for $\dot{\Omega}$ can be seen in Figure 3.2.8.

In an attempt to simplify the reward function, the survival term was removed in R_7. Comparing R_6 and R_7, we see that the general learning follows the same shape and converges at approximately the same time. As a consequence, there seems to be no benefit to adding the extra term. R_7 therefore became the final reward function used in Equation 3.2.3.

The PSF term was added to keep the agent from exclusively relying on the PSF to choose actions. Without this term, we observed an instance where the trained agent constantly set the blade pitch input u_p to -4^o and relied on the PSF to correct this. This behavior was significantly reduced by adding $r_{PSF}^{(t)}$.

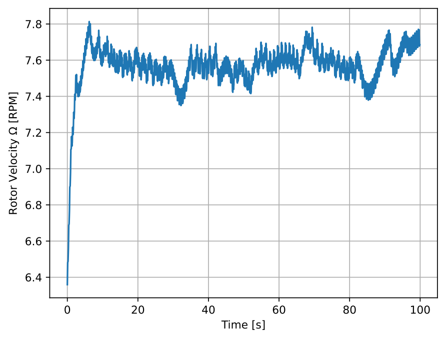


(a) Episode crash rate (smoothed).



(b) Episode reward mean.

Figure 3.2.7: Training progress for each version of the reward function displayed by episode crash rate and reward mean.



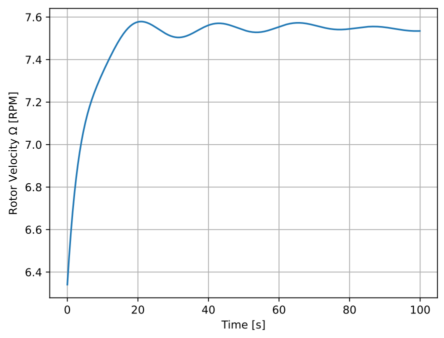
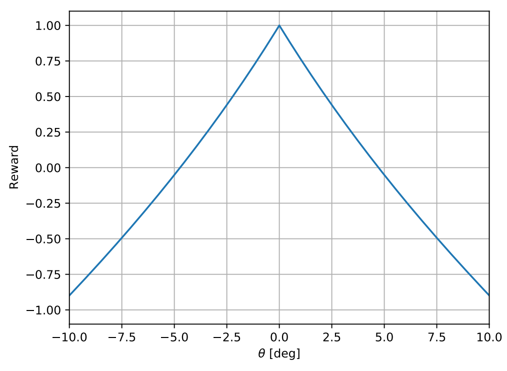
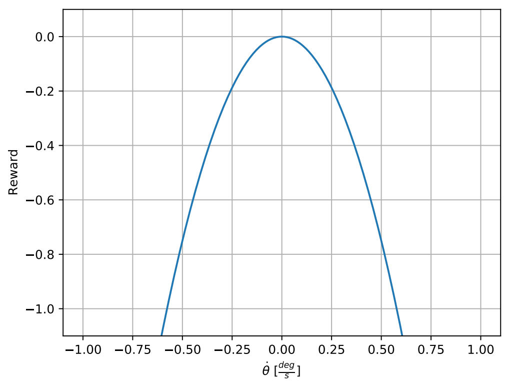


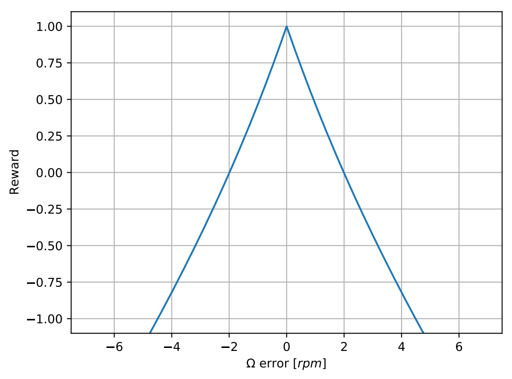
Figure 3.2.8: Response for Ω using agents trained with R_4 (left) and R_6 (right) as the reward functions.



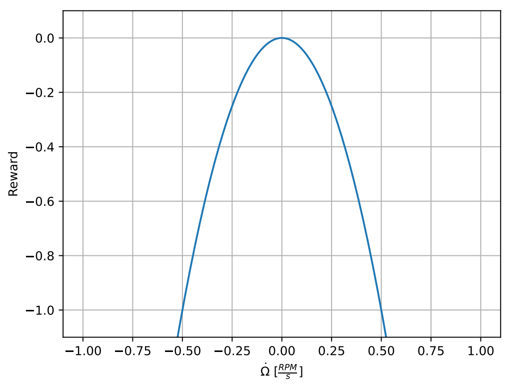
(a) $r_{\theta}^{(t)}$



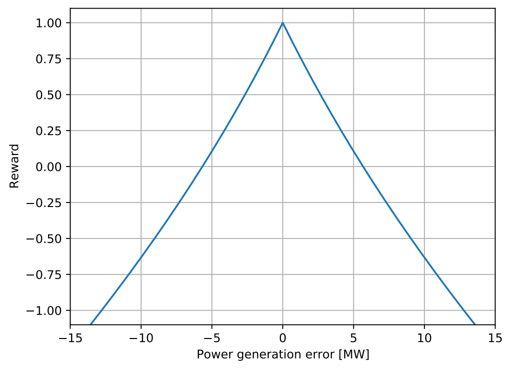
(b) $r_{\dot{\theta}}^{(t)}$



(c) $r_{\Omega}^{(t)}$



(d) $r_{\dot{\Omega}}^{(t)}$



(e) $r_{P}^{(t)}$

Figure 3.2.9: Reward curve for five first terms in the reward function.

Action and observation spaces

The action space is as described by the control inputs $[F_{thr}, u_p, P_{gen}]^T$. The control inputs were then normalized to make exploration and optimization easier for the agent. Having very different ranges for different directions of the action space might lead to one input being harder to explore than another. This can be explained by a simple example. Imagine input 1 ranges from 0 to 1000, and input 2 ranges from 0 to 1. Then an exploratory step of 0.1 would explore the action space significantly more in the input 2 direction than for input 1. Thus scaling both inputs from 0 to 1 might make it easier for the agent to learn the effects of each input.

The platform propeller thrust, F_{thr}, is scaled to range from -1 to 1, with the un-normalized range being from $-F_{thr,max}$ to $F_{thr,max}$. The value for $F_{thr,max}$ is listed in Table 3.2.5.

The rotor blade pitch, u_p, is scaled to range from -0.2 to 1, with the un-normalized range being from $u_{p,min}$ to $u_{p,max}$. The values for these are listed in Table 3.2.5.

The generator power extraction, P_{gen}, is scaled to range from 0 to 1, with the un-normalized range being from 0 to $P_{gen,max}$. The value for $P_{gen,max}$ is listed in Table 3.2.5.

Different observation spaces were explored to see what gave the best results. A natural observation to add is the state, $\mathbf{x} = [\theta, \dot{\theta}, \Omega]$, as these are the variables we want the agent to learn the dynamics of. Adding $\dot{\Omega}$ to the reward function also made it natural for it to be in the agent's observation. For the agent to be aware of its environment conditions, the wind speed was also a natural observation to include. The final observation used was $\mathbf{y} = [\theta, \dot{\theta}, \Omega, \dot{\Omega}, w_0]$. All elements in the observation are measurable with no noise.

Training

To investigate the abilities of RL in this application, we implement a framework with different versions of the environment. We choose to structure this as six scenarios, or levels, that contain wind of varying difficulty. It is assumed that the environment wind speed, w_0, is between 5 and 25 m/s. All scenarios represent wind as a slowly varying sinusoidal as defined by Equation 3.2.10. The period T_w is set to 60 seconds which is based on Figure 2.2.1 where a peak in the wind spectrum can be seen at about 60 cycles/hour, or 1 cycle per 60 seconds.

The simulated wind at timestep t is generated by the equation

$$w_0 = A \sin\left(\frac{2\pi}{T_w}t + \phi\right) + w_{mean} \tag{3.2.10}$$

where the amplitude A, wind mean w_{mean}, and wind phase ϕ are sampled randomly at the beginning of each episode by the equations

$$A = min\left(\frac{1}{2}(w_{max} - w_{min}), A_{max}\right) \cdot rand(0,1) \tag{3.2.11}$$

$$w_{mean} = (w_{max} - w_{min} - 2A) \cdot rand(0,1) + w_{min} + A \tag{3.2.12}$$

$$\phi = 2\pi \cdot rand(0,1) \tag{3.2.13}$$

where $rand(0,1)$ is a randomly generated number in the interval $[0,1]$, w_{min} is the minimum specified wind mean, w_{max} is the maximum specified wind mean, T_w is the wind period, and A_{max} is the maximum specified amplitude.

As the equations state, the phase ϕ is sampled randomly in the interval $[0, 2\pi]$ at each timestep t. Amplitude and mean value of the wind are also sampled randomly between episodes, although from different intervals depending on the level. An example of wind simulations for a level with amplitude in the 0-3 m/s range and mean in the 10-20 m/s range is shown in Figure 3.2.10. As we can see from the wind generation equations above and Figure 3.2.10, the wind speed will never go outside the specified range. A weakness of this is that larger amplitudes are not possible for wind means close to the range extremes, e.g., the combination of the amplitude of 3 m/s and wind mean of 19 m/s would not be possible for this a wind mean range of 10-20 m/s.

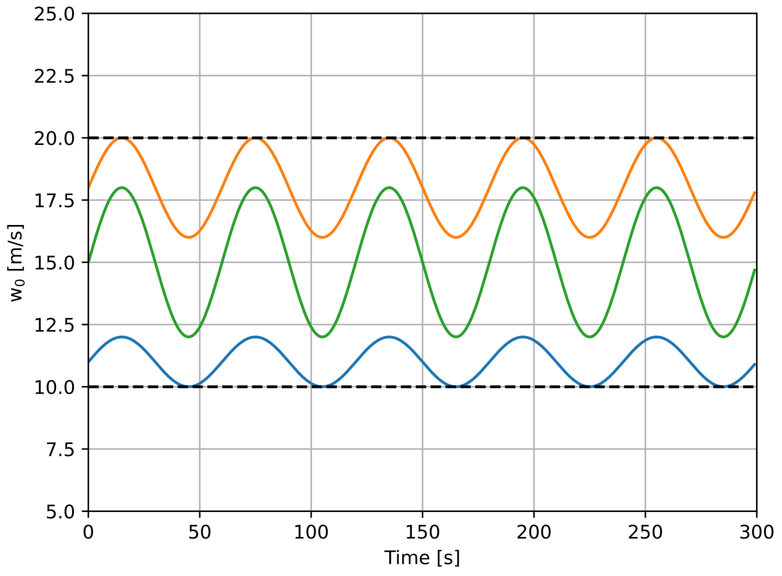


Figure 3.2.10: Examples of simulated winds in a level with amplitude in the 0-3 m/s range and wind mean in the 10-20 m/s range. Phase shift is set to zero for demonstration purposes.

Scenarios where the wind is constant were first assumed to be trivial and not realistic in real-world applications. Thus, we disregarded this special case and rather set up levels where the wind varied slowly. We add that this assumption might have been made too quickly, and more investigation should have been performed in constant wind scenarios. To aid investigation of our results, we also created Level HighWinds, Level ConstantLow, and Level ConstantHigh. These levels are further used in section 4.1. The main results are based on Level 0-5, and the extra levels are made to explore specific scenarios to aid discussion.

The levels are defined by their amplitude and mean wind sample range:

- Level 0: Amplitude 0-1 m/s. Mean 13-17 m/s.
- Level 1: Amplitude 0-1 m/s. Mean 10-20 m/s.
- Level 2: Amplitude 0-1 m/s. Mean 5-25 m/s.

- Level 3: Amplitude 0-3 m/s. Mean 10-20 m/s.
- Level 4: Amplitude 0-3 m/s. Mean 10-25 m/s.
- Level 5: Amplitude 0-3 m/s. Mean 5-25 m/s.
- (Level ConstantLow: Amplitude 0 m/s. Mean 5-13 m/s.)
- (Level ConstantHigh: Amplitude 0 m/s. Mean 13-25 m/s.)
- (Level HighWinds: Amplitude 0-3 m/s. Mean 13-25 m/s.)

The hyperparameters used for training are listed in Table 3.2.7. Parameters not listed are kept as default from Stable Baselines3. The hyperparameters were tuned using intuition and experimentation, not using a dedicated hyperparameter optimization framework. Each agent was trained for 10M timesteps.

Table 3.2.7: Non-default hyperparameters for PPO.

Hyperparameter	**Value**
n_steps	1024
learning_rate	linear_schedule(init_val=1e-4)
gae_lambda	0.95
gamma	0.99
n_epochs	4
clip_range	0.2
ent_coef	0.01

Implementing RL in a specific application such as this requires being able to view data from, and get insight into, the training process. Thus, a great effort has been put into saving as much data as possible to debug and evaluate the performance of agents. The current solution implements this through callbacks, reporting to Tensorboard, and saving data to files during training.

Performance and testing

The performance of each agent was tested by running it in 100 randomly generated episodes. For training performance, this was done in the level it was trained in. The maximum number of timesteps for each episode was also increased from 3000 to 6000 when testing to investigate if the agents are exploiting the maximum timestep limit or are able to extend to continual operation.

The metric for performance is based on the cumulative reward and created such that it reflects how close to the theoretical maximum of this the agent is. Performance is set to be the cumulative reward for the episode, excluding r_{PSF}, divided by the theoretical maximum. It is then multiplied by 100 to make it resemble a percentage measure. r_{PSF} is excluded to make the metric comparable between PSF-based and non-PSF agents. Crashes are considered as a separate metric, and as a result, we exclude the episodes that crashed from the average, i.e., the performance is averaged between the episodes that did not crash. Including crashes in the metric would make it hard to separate if the metric is low because of crashes or because of low control performance. In addition, including crashed episodes in the performance metric would lead late-crashing episodes to lower the metric less than early-crashing ones, which is undesirable because a crash is considered a crash no matter when it happens. This setup is also convenient when comparing the RL performance to the performance of the RL with PSF, which optimally should eliminate crashes altogether.

The resulting performance metric for episode i is then defined as

$$(Performance)_i = 100 \cdot \frac{(\text{Cumulative Reward})_i - (\text{Cumulative PSF Reward})_i}{\text{Theoretical max cumulative reward}} \tag{3.2.14}$$

In our case, we run 100 episodes to get a better estimate of the true performance. The resulting average performance is then

$$Performance = \frac{1}{M}\sum_{k=1}^{M}(Performance)_k \tag{3.2.15}$$

where $M = 100 -$ (Number of episodes that crashed). Even though Performance is meant to represent a percentage of the maximum, we emphasize that this theoretical maximum might not be feasible for the system, especially not in all wind conditions. The testing uses the same performance measure.

As mentioned, excluding the crash rate from the metric helps us separate crash rate and low control performance as two different metrics. However, it can also create misleadingly high performance for an agent with a high crash rate. We, therefore, stress that as long as the crash rate is non-zero, the results for performance are not as valid and stay inconclusive. In any case, it always has to be looked at in combination with the corresponding crash rate.

Another interesting result is to see how much the RL agent is corrected by the PSF after it is trained. We display this PSF metric as a percentage of the minimum PSF reward for an episode. Minimum PSF reward is derived from Equation 3.2.9, and gives $-5(2 + 1.2 + 1) = -21$ for each timestep. This comes out to $-21 \cdot 6000 = -126000$ for each episode. This is then averaged between 100 episodes. In the same way, as with the performance metric, the episodes that crashed are excluded from the average.

$$\begin{aligned} (PSF\ metric)_i &= 100 \cdot \frac{\text{Cumulative PSF Reward}}{\text{Theoretical min. cumulative PSF reward}} \\ PSF\ metric &= \frac{1}{M}\sum_{k=1}^{M}(PSF\ metric)_k \end{aligned} \tag{3.2.16}$$

where $M = 100 -$ (Number of episodes that crashed).

3.2.4 The terminal set

To express the terminal set, we need the Jacobian $\nabla^{\intercal}\mathbf{f}$ of our system, which is calculated to be

$$\nabla^{\intercal}\mathbf{f}_{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 \\ \Theta(\theta,\dot{\theta}) & C_3\cos(x_1) & C_5\frac{\partial F_{wind}}{\partial x_3} \\ 0 & 0 & \frac{1}{J_r}(\frac{\partial Q_{wind}}{\partial u_3} + \frac{x_3}{x_3^2}) \end{bmatrix} \tag{3.2.17}$$

$$\Theta(\theta,\dot{\theta}) = C_1\cos(2x_1) + C_2\cos(x_1) - C_3\sin(x_1)x_2 \tag{3.2.18}$$

$$\frac{\partial F_{wind}}{\partial x_3} = k_r(\cos(u_2)w - 2\sin(u_2)\ell x_3) \tag{3.2.19}$$

$$\frac{\partial Q_{wind}}{\partial x_3} = -k_r\sin(u_2)\ell w - 2b_r x_3 \tag{3.2.20}$$

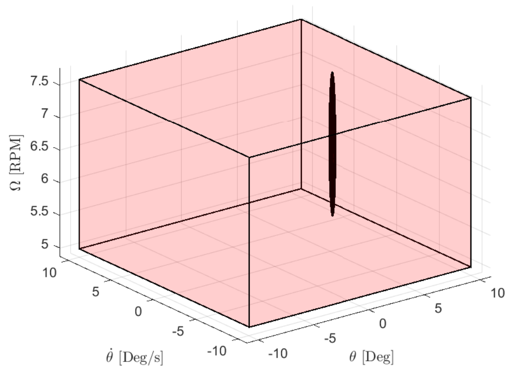
$$\nabla^{\intercal}\mathbf{f}_{\mathbf{u}} = \begin{bmatrix} 0 & 0 & 0 \\ C_4 & C_5k_r(\sin(u_2)wx_3 - 2\cos(u_2)\ell x_3^2) & 0 \\ 0 & \frac{-1}{J_r}(k_r(\sin(u_2)w^2 + \cos(u_2)\ell w x_3) & \frac{1}{J_r x_3}) \end{bmatrix} \tag{3.2.21}$$

Inspecting the Jacobian and the system equation displayed in Equation 3.1.7, we see that $p = 9$. Reformulating the system as a function of constants and $\boldsymbol{\delta} = [\delta_1, ..., \delta_p]^{\intercal}$, we obtain

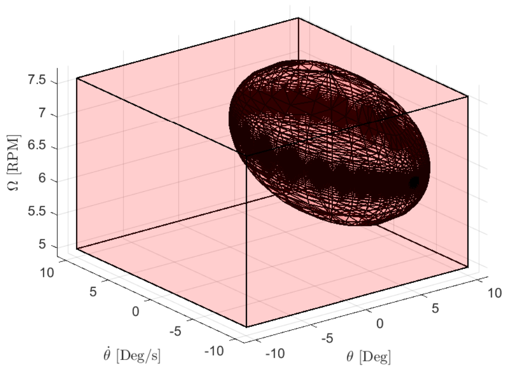
$$\dot{\tilde{\mathbf{x}}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \delta_1 & \delta_2 & \delta_3 & \delta_5 \\ 0 & 0 & \delta_4 & \delta_6 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tilde{\mathbf{x}} + \begin{bmatrix} 0 & 0 & 0 \\ C_4 & \delta_7 & 0 \\ 0 & \delta_8 & \delta_9 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{u} \quad , \tag{3.2.22}$$

$$\boldsymbol{\delta} \in \boldsymbol{\Delta} = \bigtimes_{i=1}^{10} [\underline{\delta_i}, \overline{\delta_i}] \tag{3.2.23}$$

Casadi was used to derive the Jacobian and automate the derivation above.



(a) The true terminal set.



(b) The alternative terminal set.

Figure 3.2.11: The different ellipsoidal sets plotted against the constraints in the PSF. The red box defines the constraints while the black ellipsoid represents the terminal set.

The constraints in Equation 3.1.13 were tightened to create less uncertainty in $\mathbf{A}(\boldsymbol{\delta})$ and $\mathbf{B}(\boldsymbol{\delta})$. The changed constraints for system uncertainty can be found in Table 3.2.8.

Table 3.2.8: Constraints used for obtaining system uncertainty.

Variable	Lower Constraint	Upper Constraint	Unit
Ω	6	7	[rpm]
θ	0	8	[deg]
$\dot{\theta}$	0	0	[deg/s]

The resulting terminal set is quite restricted in size due to a combination of our system dynamics, the inherent linear conservatives of the method and input constraints. The ellipsoidal terminal set can be seen in Figure 3.2.11a. If the states are far from the terminal set, the PSF requires a long horizon to drive the state into this set. Keeping the PSF timestep $\tau_k = \tau_t$ relatively small (<1 s), for reasons we will addressed in subsection 3.2.5, leads to a large number of optimization variables $(n_x + n_u) * N$. Not only does the increased complexity of the problem lead to slow evaluation, it was observed to also lead to higher failure rates in the solver.

The addition, the PSF filter will significantly lower the framerate of the RL training process. With a naive approach, we were only able to achieve 20 evaluations per second during training on 12 CPU cores. This is in contrast to 1500 using just RL agent. The RL method is dependent on high framerate during training, and to address the issue a wide variety of actions were taken and explored.

Firstly, and perhaps most drastically, we created a new terminal set. The new set is also an ellipsoidal one, but was created by simply maximizing it within the state constraints, negating the system dynamics and input constraints. The result can be seen in Figure 3.2.11b. Obviously, we no longer have a strong proof related to the stability of the system. While this seems to negate an important aspect of why we introduced the PSF, we will further examine the impact of this simplification in section 4.1.

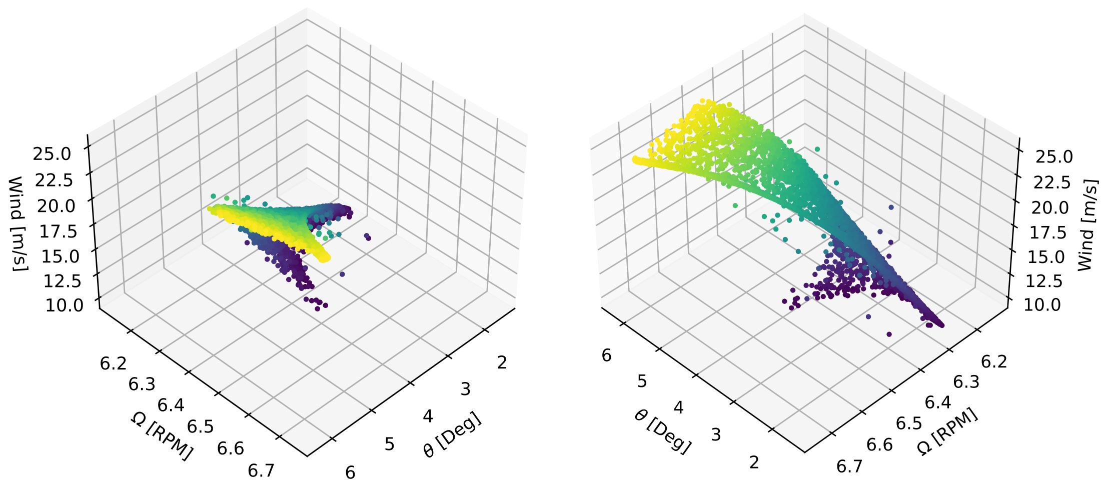


Figure 3.2.12: Steady state plane viewed at different angles. Colored by wind speed, blue is low and yellow is high.

As some of our control inputs have a large numerical range where one input is given in 10^6 and one as small as 10^{-1}, we also need to scale the input to be able to solve the terminal set and our receding horizon problem. This is related to the same issue as when scaling the action space of the RL agent. For the PSF, this was handled by simply scaling $\mathbf{B}(\boldsymbol{\delta})$ by the max range of the polytopic constraints similarly to Equation 3.1.13. This leads to columns of $\mathbf{B}(\boldsymbol{\delta})$ being scaled, which also scales the columns of the polytopic constraints. The rows in $[\mathbf{H_z}, \mathbf{h_z}]$ were normalized further with a constant, such that the Euclidean sum of each row was 1. Without these normalizations, MOSEK was not able to find an adequate solution. The algorithm of obtain the terminal set is listed in Appendix A.

3.2.5 Predictive safety filter implementation

As each state and input introduces added complexity to our problem, we chose not to include Equation 3.1.6 in our PSF formulation. Instead, the even simpler approximation $w \approx \frac{2}{3}w_0$ was used. The argument for the simplification is done on the basis of assuming $\dot{\theta}$ sufficiently small. As mentioned subsection 3.2.5, we normalized the differences in scale between the inputs in the PSF's objective function. This was done by setting $\mathbf{R}$ in the same manner as Equation 3.1.15. The scaling has significant importance in our case as the differences are so big. The way we shaped our PSF cost function Equation 3.1.8 was also observed to improve the solver stability.

Due to low solver speeds, the number of time steps in the PSF horizon was set to $N = 20$. A longer horizon gave diminishing returns with regards to the added computational complexity. In combination with the alternate terminal set, the discretization step length for the PSF was chosen to be identical to the RL simulation timestep for the initial PSF timestep, i.e. $\delta_0 = 0.1$. For the rest of the horizon the timestep was set to 0.5 - leaving a temporal span of $T = 10.1$s. The prediction horizon was observed to be sufficient to enter the terminal set for most of the simulations. To realize the discretization function $\Phi(\cdot)$, we used the simpler RK4 rather than the method in Table 3.2.1. The PSF sample time δ_{t_i} was varied in an attempt to prolong the PSF's temporal horizon, where the δ_{t_i} increases towards the end of the horizon. It was observed not to have a significant impact and led to more solver errors. In general, it was observed that even for one-step prediction, a sampling time above 1s was not able to run properly. The understanding was that our discretization method was not stable for large time steps. We will discuss in greater detail in section 4.2 due to its importance for the general progression and result of this report.

We tighten the constraints from the RL agent to the constraint given in Table 3.2.2. This was done to attain the desired operation range given by Gaertner et al. (2020) for the RWT. Furthermore, we observed that the differences between the approximated model in the PSF and the RL model created the need for some leeway before the episode was deemed as crashed in the simulation. Of course, this means that a more complex control problem is handed to the PSF than the RL agent, but this is further addressed in section 4.2. Moreover, we constrained the inputs in the PSF with saturation limits, which can be seen in Table 3.2.9.

Table 3.2.9: Input rate saturations in the PSF.

Variable	Lower Saturation	Upper Saturation	Unit
$\dot{u}_p$	-8	8	[deg/s]
$\dot{F}_{thr}$	$-1.5 \cdot 10^5$	$1.5 \cdot 10^5$	[N/s]
$\dot{P}_{gen}$	$-5 \cdot 10^6$	$5 \cdot 10^6$	[W/s]

To further increase the PSF evaluation speed, a few more changes were applied. As most nonlinear optimization solvers, IPOPT requires an initial guess on the solution to evaluate the first gradient. The default value is 0, but as mentioned previously, this lead to problems due to numerical issues as a result of zero division. At the initialization of a new episode, all optimization variables were initialized to their steady-state solution as given by Equation 3.1.14. The subsequent ones are given by the previous admissible solution from the solver, known as a warm-start strategy. This added beneficial speed-ups but also adds to the risk of solver infeasibility and local convergence. An increase in the aforementioned problems was in fact observed but the change was still deemed as necessary. The resulting PSF evaluation speed moved from 20 to 160 evaluation per second on 12 cores as a consequence of implemented speed-ups.

Chapter 4

Results and Discussions

In this chapter, we present and discuss the results of our work. The results will serve as exemplifying use cases for our framework, which will enable further discussion. First, we address the results from the agents' training, then move on to generalization results by testing the agents in different levels. We also look at the system response to see the qualitative results for the agent-controllers before summarizing the results with an overview. The discussion will follow the same structure but with a greater focus on the RL-PSF combination. We first aim to discuss the result directly given our model before moving on to possible limitations and error sources. We will also evaluate their importance and impact on our result, paving the way for further improvements and the framework's general outlook. For the sake of clarity, we also refer to the combined control scheme of an RL agent and a PSF as the combined controller. In addition, the process of applying the PSF to the RL control actions may be referred to as simply filtering.

4.1 Results

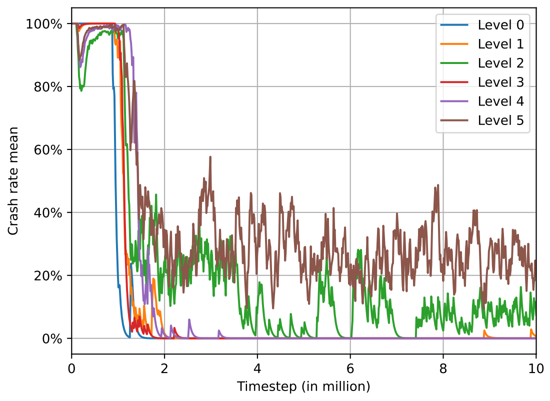
In this section, we present the results of our study. In particular, we focus on the progress and performance of agents during the training, test these agents in other levels, and lastly the qualitative response of the system with the RL-PSF controllers. The pure RL implementation serves two purposes. Firstly, it is used to generate results which are used for benchmarking and secondly, it gives a foundation for the implementation of the PSF.

In plots marked with "(smoothed)", an exponential moving average smoothing with a weight of 0.8 was applied to make the plots more legible.

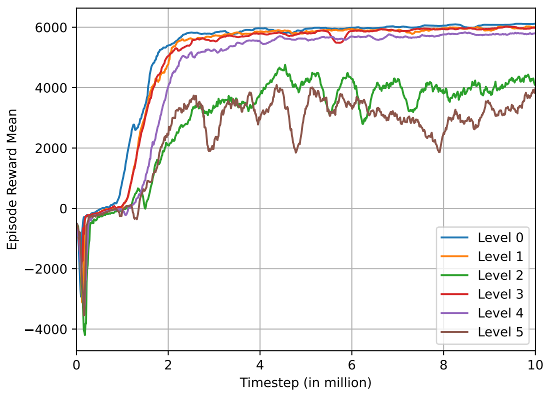
4.1.1 Training

We first focus on the training of the RL agent in the absence of a PSF, then shift the focus to include the PSF.

Training without the predictive safety filter



(a) Episode crash rate (smoothed).



(b) Episode reward mean.

Figure 4.1.1: Training progress displayed by episode crash rate and reward mean.

The agents were trained for each level 0-5 without any intervention from the PSF. Figure 4.1.1 shows the training progress of each agent in terms of the crash rate and episode reward. We see that the agents demanded a significant number of timesteps to learn the dynamics, but most of them converged after about 3 million. Figure 4.1.1b shows that Agent 2 and 5 fail to converge to a clear optimum, and Figure 4.1.1a shows a crash rate of about $10 - 40\%$, even after 10M timesteps. For Agent 0, 1, 3, and 4, the rewards converged at about 3M timesteps, which implies that the agents found their respective optima. However, they all converged to slightly different values as the wind conditions got more challenging for higher levels and high rewards became less obtainable.

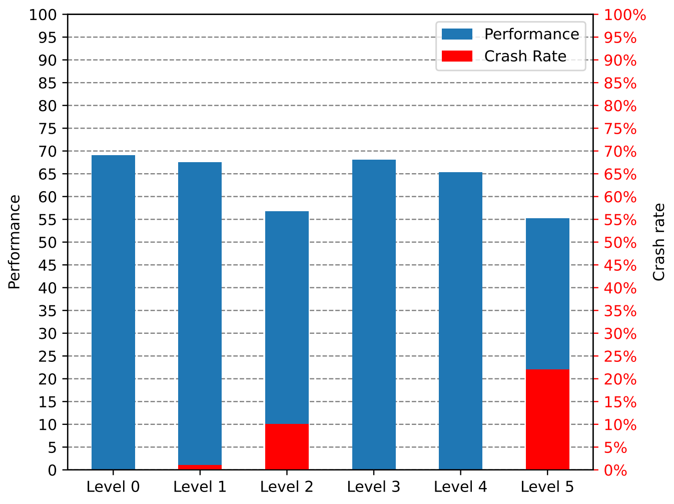
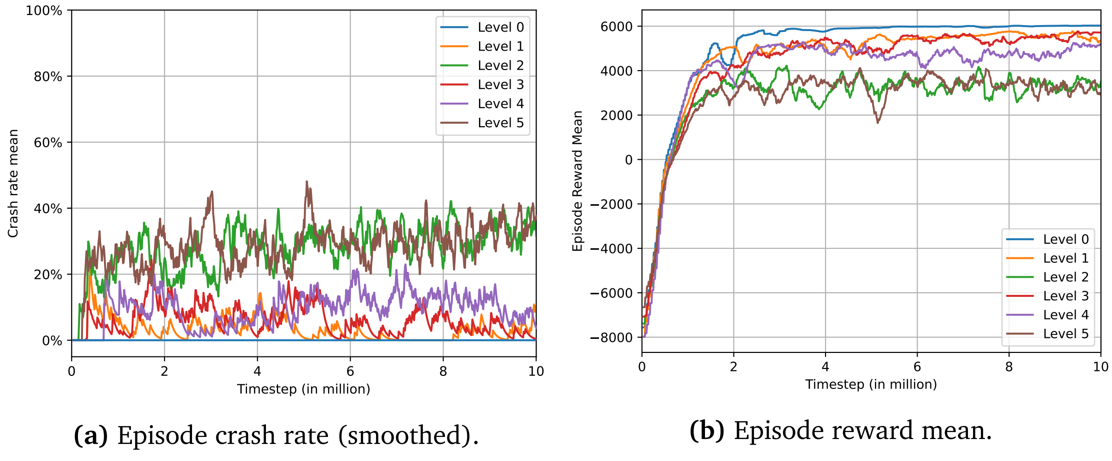


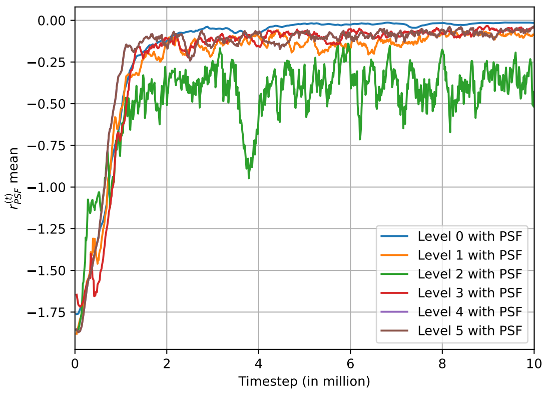
Figure 4.1.2: Results for performance of agents trained at each level.

In addition to looking into what happens during the training, we wanted to investigate the performance of agents after the training. To see how well the agents were able to learn the system dynamics and apply a valid control strategy, we test each agent's performance at the same level it was trained on. The performance is averaged between 100 episodes of up to 6000 timesteps each as explained in section 3.2. The results are shown in Figure 4.1.2. These numbers confirm the results indicated by the training progress in Figure 4.1.1. The performances for Agent 2 and 5 are noticeably lower than the others while also having higher crash rates. Agent 0, 1, 3, and 4 all perform relatively similarly, although we note the non-zero crash rate of 1% for Agent 1.

With predictive safety filter



(a) Episode crash rate (smoothed). **(b)** Episode reward mean.



(c) Timestep PSF reward mean (smoothed).

Figure 4.1.3: Training progress for agents trained with a PSF displayed by episode crash rate and reward mean. Mean PSF reward per timestep (smoothed) is also included.

One of the objectives of this thesis is to investigate the effect filtering has on the training. Therefore, the same agents mentioned in the previous section were retrained but this time with the PSF. The training progress can be seen in Figure 4.1.3. The results here are quite different when compared to Figure 4.1.1, and we observe that Agent 0 is the only one that seems to converge satisfactorily and have zero crashes during the training. In other words, the PSF fails to prevent crashes during training and actually has a negative impact on convergence for levels 1-5. For Level 0, on the other hand, we observe more desirable results with successful crash prevention and convergence to an optimum. Similarly as for RL without the filter, Level 2 and 5 seem to be the hardest for the combined controller as well. Figure 4.1.3c shows that the agents are able to learn to make safer actions as the PSF reward penalty gets smaller as training goes on. Again, Agent 0 is the one with the best results here with a PSF penalty of *close to* zero at the end of training. It is worth noting that there was a significant slow down in the framerate with the PSF, about 140 fps versus the 1500 fps without filtering.

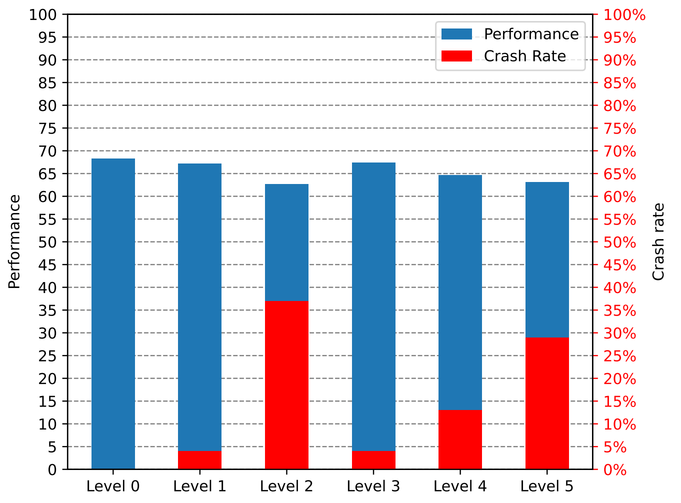


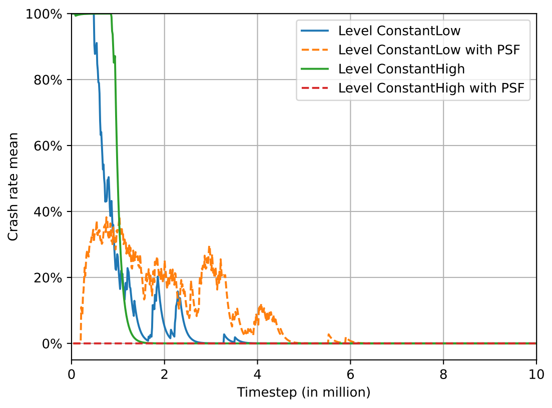
Figure 4.1.4: Results for performance of agents trained with filtering at each level.

Looking at the performance of the trained agents in Figure 4.1.4, we see that it is very similar for all levels. The performance for Agent 2 and 5 is still lower than for the other levels, but it is increased from Figure 4.1.2. However, we make another important observation; The crash rate is significantly increased for all levels except Agent 0. This is the opposite of the desired effect, and it is clear that the PSF struggles to find correct solutions for the system in the conditions in Level 1-5.

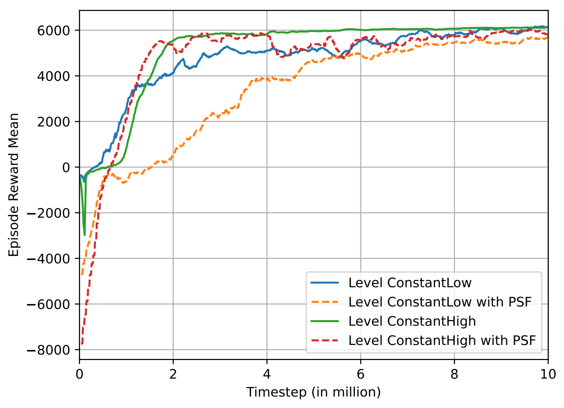
To investigate these results further, we need to look at the differences between the levels. The seemingly most challenging levels (2 and 5) are the only ones containing winds down to 5 m/s, and Level 0 is the only level that does not contain wind speeds below 13 m/s. This, together with the fact that Level 0 is the only level the PSF is able to prevent crashes for, leads us to believe that problems occur in low wind speeds (<13m/s). There is also a possibility that it is the variation in the wind that is causing the problems. Three new levels were created to address this: Level ConstantLow consisting of constant wind in the 5-13 m/s range, Level ConstantHigh consisting of constant wind in the 13-25 m/s range, and Level

HighWinds consisting of varying winds with a mean wind speed in the 13-25 m/s range and an amplitude in the 0-3 m/s range.

Comparison with and without predictive safety filter for constant winds



(a) Episode crash rate mean, with and without filtering (smoothed).

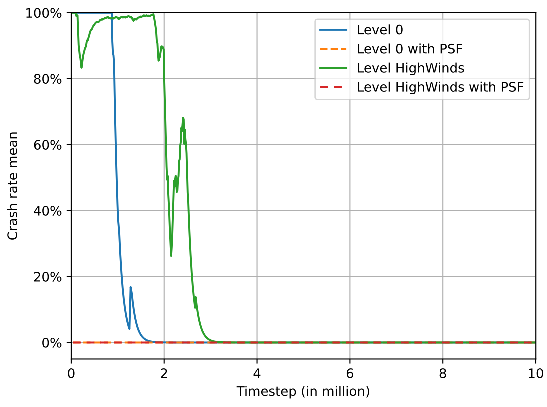


(b) Episode reward mean with and without filtering.

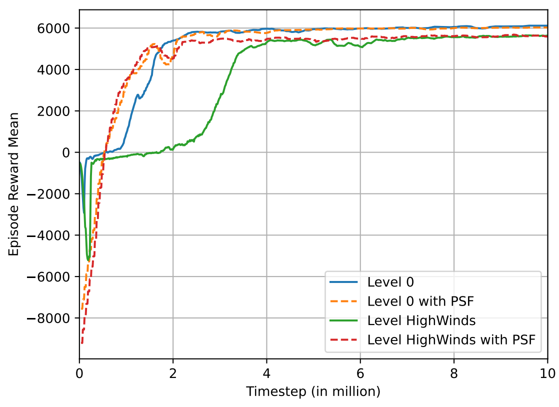
Figure 4.1.5: Training progress for agents in Level ConstantLow and ConstantHigh.

Figure 4.1.5 shows the training progress for the levels with constant wind. As our hypothesis suggests, the PSF successfully prevents crashes during training in Level ConstantHigh, while it fails to do so for winds below 13 m/s in Level ConstantLow. This strengthens our hypothesis of low winds being the problem. However, we note that the RL agent alone is actually able to learn how to prevent crashes and seems to converge for both low and high winds when they are constant.

Comparison with and without predictive safety filter for varying high winds



(a) Episode crash rate mean, with and without filtering (smoothed).



(b) Episode reward mean, with and without filtering.

Figure 4.1.6: Training progress for agents in Level 0 and Level HighWinds.

Figure 4.1.6 shows the training progress for Level HighWinds. Level 0 is also added for reference. The most important thing to note is shown in Figure 4.1.6a; The PSF was in fact able to prevent crashes for the entirety of training in Level HighWinds. Together with the results for constant wind, this confirms our hypothesis that low wind speeds are a problem for the PSF. It also indicates that varying winds are not as much of a problem.

Because the PSF did not perform satisfactorily for Level 1-5, a focus is put on Level 0 and Level HighWinds when comparing performance and training progress to the agents without filtering. Because the performance metric always has to be considered with the crash rate in mind, it becomes a more interesting metric once the crash rate is zero. From Figure 4.1.6 we see that there is a clear difference in training progress with and without the PSF, especially in the learning curvature. For Level 0, the learning is not notably accelerated, and the two agents converge at approximately the same time. However, this changes when changing the wind conditions. For Level HighWinds we see a clear improvement in the learning speed as the agent without the PSF takes almost twice as many timesteps to converge. This is accomplished while having no crashes, as opposed to the pure RL agent crashing every episode in the initial part of the training.

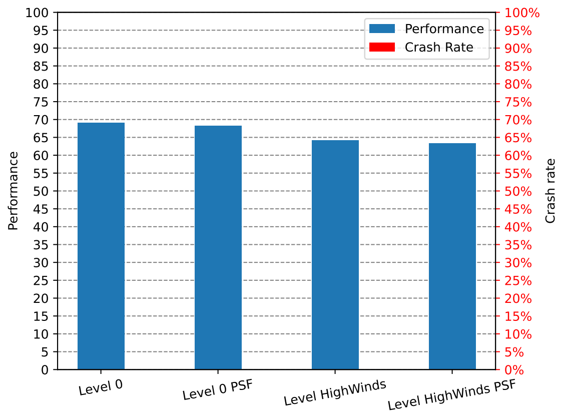
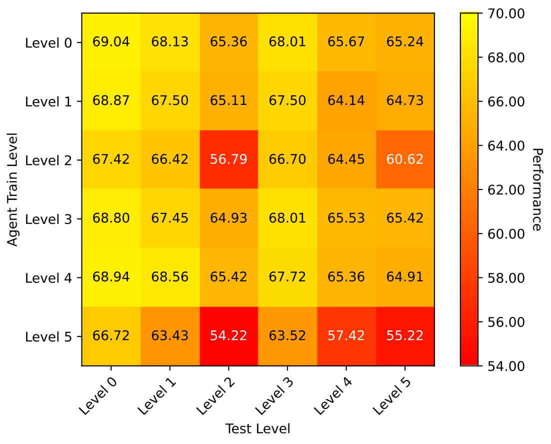


Figure 4.1.7: Performance of agents with and without filtering for Level 0 and HighWinds.

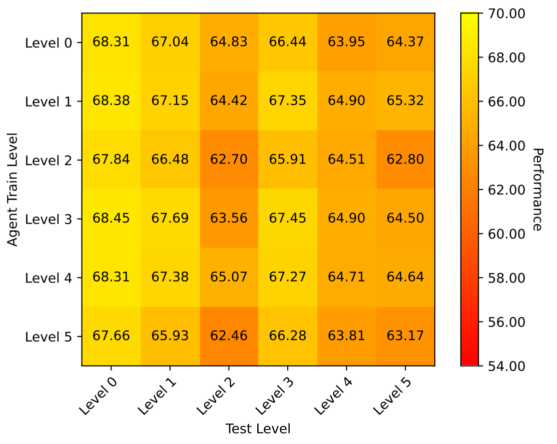
Figure 4.1.7 confirms the findings in Figure 4.1.6 and shows that both agents trained with and without filtering are able to prevent crashes for varying winds above 13 m/s. The figure also shows no significant difference in performance by adding the PSF. This is when tested at the same level as the agents were trained in.

4.1.2 Testing

The test results are presented as heatmaps in Figure 4.1.8 and 4.1.9. On the vertical axis, we have the train level of the agent, while on the horizontal axis is the level in which that agent was tested. This means the top row contains values for how the agent trained in Level 0 (Agent 0) performed in all the different levels and similarly for the others. We also refer to these results as generalization results because running agents in the other scenarios gives an indication of its generalization performance.

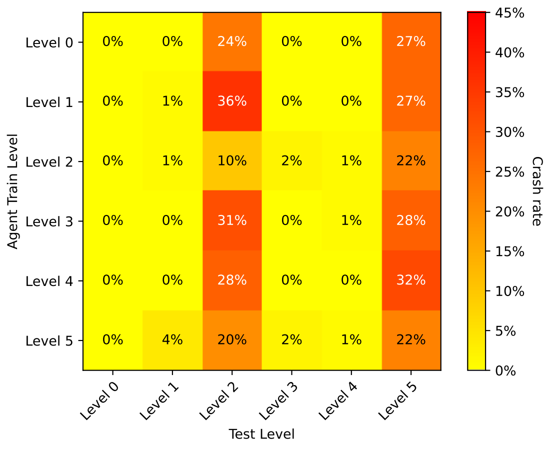


(a) Without the PSF

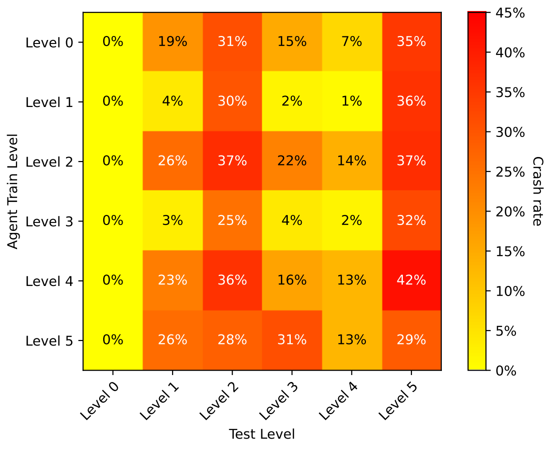


(b) With the PSF

Figure 4.1.8: Generalization performance. Higher is better.



(a) Without the PSF



(b) With the PSF

Figure 4.1.9: Generalization crash rate. Lower is better.

Without predictive safety filter

Focusing on Figure 4.1.9a we see that all agents had a significant crash rate in Level 2 and 5. This indicates that these two levels were the hardest to achieve optimal control in, at least in the eyes of the RL agent, which is consistent with the training results. In terms of crash rate, Level 0 was the least demanding level, and all agents were able to complete 100 episodes without crashing here. All other levels had one or more agents crash during the test episodes. Even Level 1, which is meant to only be slightly more demanding than Level 0, had three out of six agents crash in at least one episode. Figure 4.1.9a does however show that the crash rates are low and the agents are close to learning the optimal control strategy for Level 0, 1, 3, and 4.

In terms of performance in Figure 4.1.8a, none of the agents stand out as noticeably better than all others. However, Agent 2 and 5 stand out as especially poor again. Table 4.1.1 shows that Agent 0 actually had the best average performance overall levels. Again Agent 2 and 5 are separated from the others. Another interesting result in Figure 4.1.8a is that Agent 0 performs better in all levels compared to the respective agent trained in that level.

Table 4.1.1: Average performance and crash rate over all levels for each agent trained without filtering.

Agent Level	Avg. Performance	Avg. Crash rate
0	66.91	8.50%
1	66.31	10.67%
2	63.73	6.00%
3	66.69	10.00%
4	66.82	10.00%
5	60.09	8.17%

With predictive safety filter

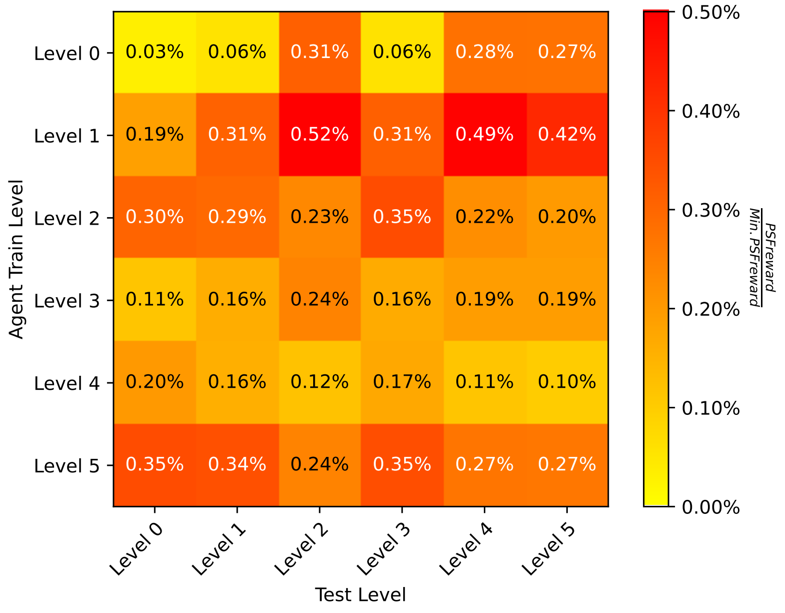
From Figure 4.1.8 and 4.1.9 it is clear that the PSF has an impact on both performance and crash rate. In terms of performance, the PSF seems to improve performance for the lowest-performing agents without filtering. Table 4.1.2 confirms this but also shows that filtering has a negative impact on the other, well performing, agents and performance are slightly degraded.

The crash rate, however, tells a different story. The agents trained with the PSF actually have overall higher crash rates than the ones trained without. The elevated number of crashes is especially prominent for the levels that the pure RL agents performed well in. Figure 4.1.9b also confirms our findings in the training results. Level 0 is again the only level where the PSF successfully guarantees crash prevention for all agents.

Table 4.1.2: Average performance and crash rate over all levels for each agent trained with filtering.

Agent Level	Avg. Performance	Avg. Crash rate
0	65.82	17.83%
1	66.25	12.17%
2	65.04	22.67%
3	66.09	11.00%
4	66.23	21.67%
5	64.89	21.17%

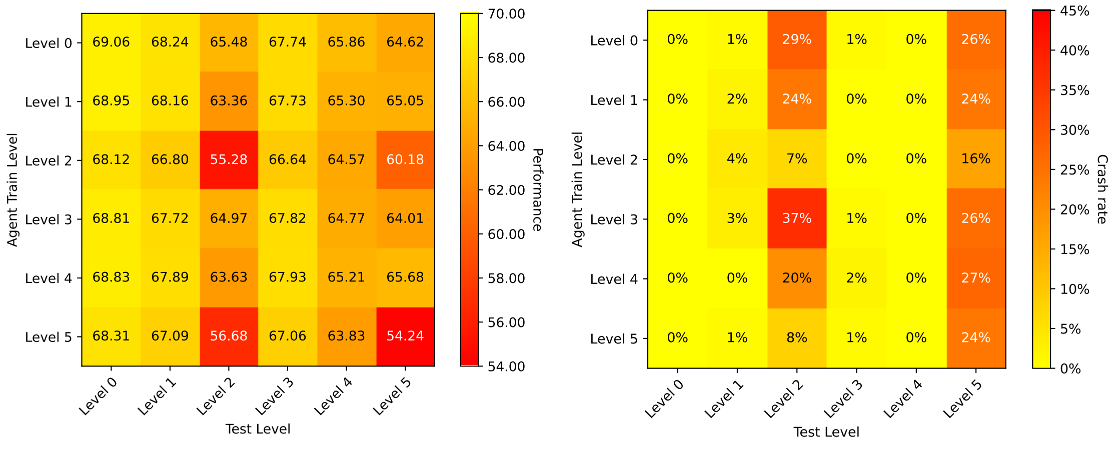
Using the PSF metric in Equation 3.2.16, Figure 4.1.10 shows the average percentage of theoretical minimum PSF reward for each agent in each level. We observe that the amount of correction from the PSF is low, staying in the range of $0.03\% - 0.52\%$. However, it is important to note that 100% represents the extreme where the agent proposes actions are at one extreme of the action space, and the PSF corrects it to the total opposite extreme. Figure 4.1.3c shows that the actual lowest observed mean PSF reward is about -2 per timestep, which turns out to about 9.5%. So the numbers have to be considered with that in mind. The numbers do not show a clear pattern but Agent 0 is again one of the best performing with Agent 4 also showing decent results.



	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Level 0	0.03%	0.06%	0.31%	0.06%	0.28%	0.27%
Level 1	0.19%	0.31%	0.52%	0.31%	0.49%	0.42%
Level 2	0.30%	0.29%	0.23%	0.35%	0.22%	0.20%
Level 3	0.11%	0.16%	0.24%	0.16%	0.19%	0.19%
Level 4	0.20%	0.16%	0.12%	0.17%	0.11%	0.10%
Level 5	0.35%	0.34%	0.24%	0.35%	0.27%	0.27%

Figure 4.1.10: Avg. percentage of theoretical minimum PSF reward for each trained agent in each level. The metric is defined in Equation 3.2.16.

Next, we look at the performance and crash rate if we have an agent trained without filtering and then deploy it with a PSF. The results are shown in Figure 4.1.11. The results are overall very similar to deploying without a PSF (Figure 4.1.8a and 4.1.9a). This indicates that the PSF neither degrades nor improves performance once the agent is trained and deployed. As clearly shown by previous results, the PSF is not able to prevent crashes for levels 1-5.



(a) Performance **(b)** Crash rate

Figure 4.1.11: Generalization performance and crash rate for agents trained without filtering but deployed with a PSF.

Lastly, we investigate the generalization performance in the successful scenarios by testing Agent 0 in Level HighWinds, both with and without filtering. The results can be seen in Figure 4.1.12, where we observe that the performance is noticeably lower in Level HighWinds than in the level the agent was trained in. This is as expected and the difference is still sufficiently small for us to consider the agent to successfully generalize to a wider range of winds an larger wind amplitudes than it was trained on, here from 13-17 m/s to 13-25 m/s and from 0-1 m/s to 0-3 m/s.

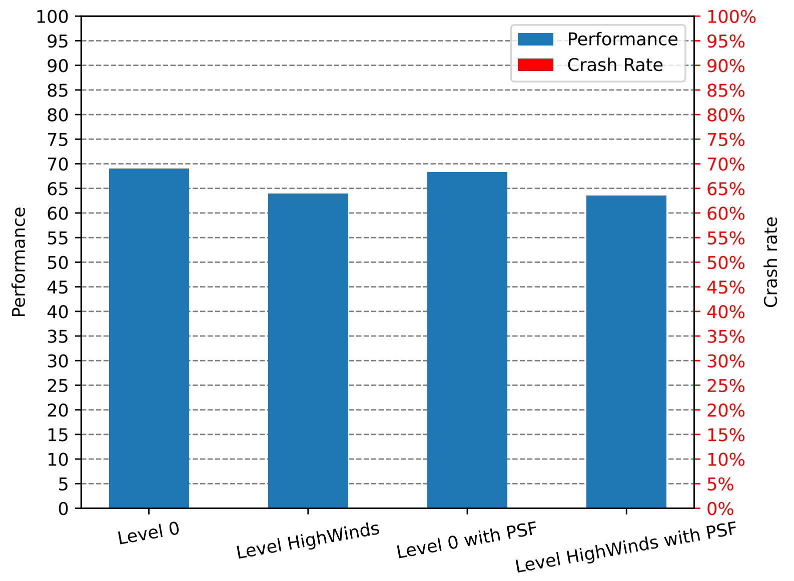
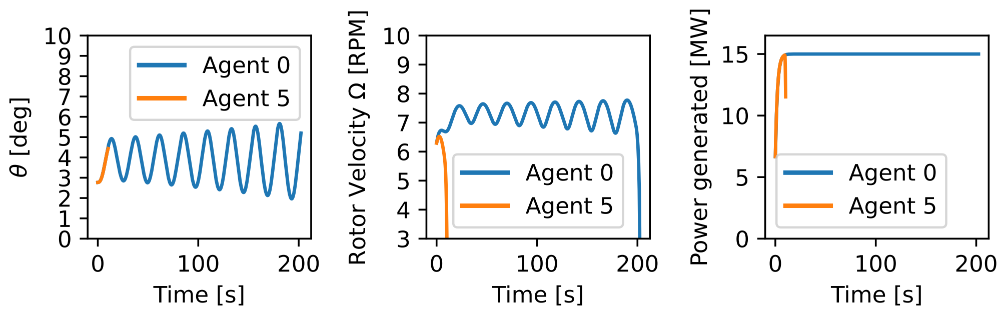


Figure 4.1.12: Performance of Agent 0 in Level HighWinds. Performance in Level 0 is added for reference.

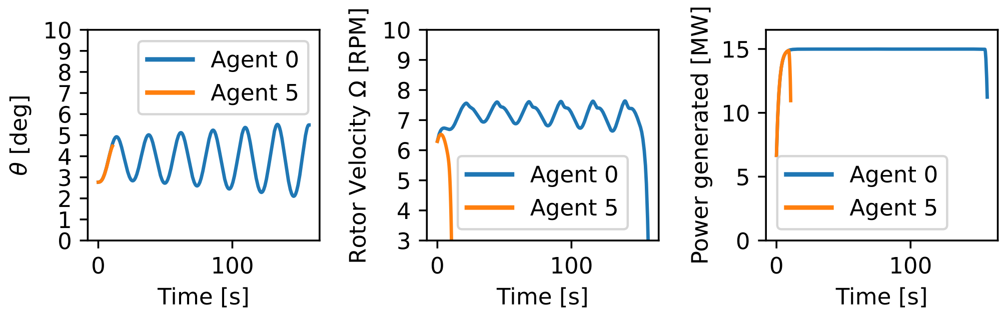
4.1.3 System response

Agent 0 and 5 are chosen for comparison of qualitative response since these are the overall best and worst-performing out of the six main agents. We look at the response of agents trained without a PSF but then compare these when deployed with and without filtering.

To further investigate our hypothesis of bad performance for low wind speeds, we simulate the agents in a scenario with a constant wind of 10 m/s. The response can be seen in Figure 4.1.13. Both agents crash, and it is clear that these system responses are undesirable. Deploying the agents with a PSF does not seem to improve this. While the response is poor for both agents, we observe that Agent 0 is able to keep the turbine from crashing for longer than Agent 5, even if Agent 5 is trained in low winds and Agent 0 is not.



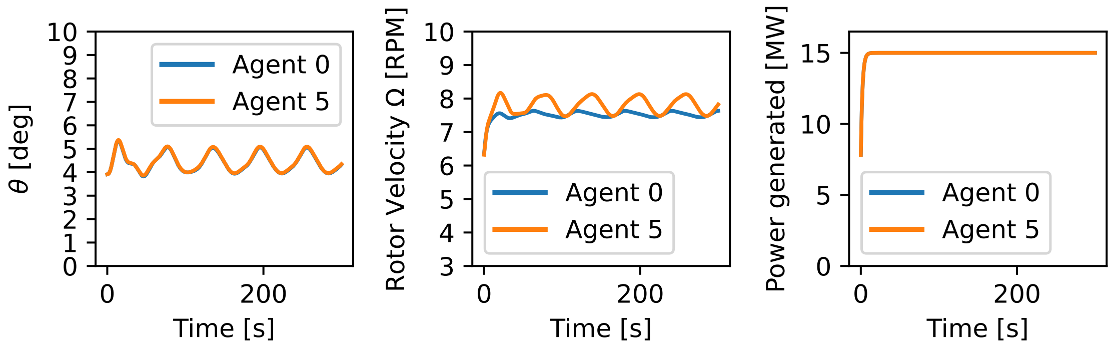
(a) Response for constant wind of 10 m/s, agent deployed without filtering.



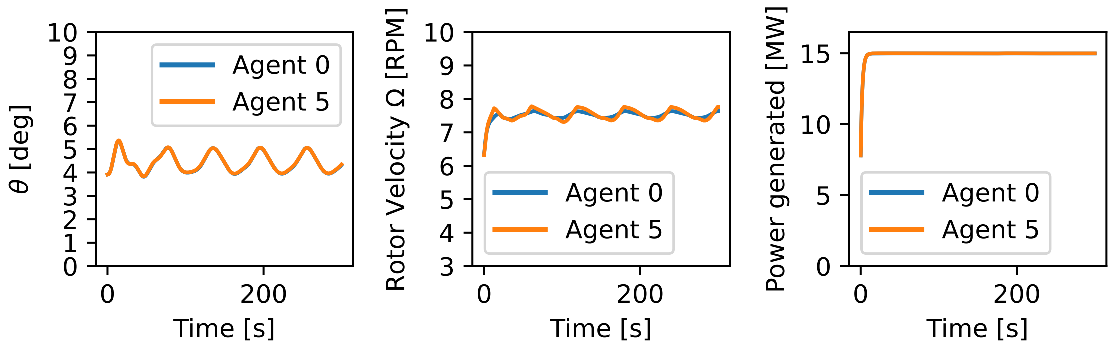
(b) Response for constant wind of 10 m/s, agent deployed with PSF.

Figure 4.1.13: Response of system in constant low wind, with and without filtering.

For our main comparison in the response of agents deployed with and without filtering, we keep the wind speeds above 13 m/s because results in subsection 4.1.1 and 4.1.2 indicate crashes for low winds. This is also exemplified by Figure 4.1.13. Comparing Figure 4.1.14a to 4.1.14b and Figure 4.1.15a to 4.1.15b we observe a clear improvement in the response of Ω for Agent 5 when deploying with a PSF. The PSF intervenes to attempt to keep the rotor speed within the constraints set in the PSF (Table 3.2.2), and while the system still goes slightly outside the constraints, there is a significant improvement. The PSF does not seem to intervene for Agent 0, which is expected because the agent nearly keeps the system within constraints by itself. θ is virtually identical for Agent 0 and 5, both with and without filtering. This indicates that the platform thruster does not have much impact on the system.

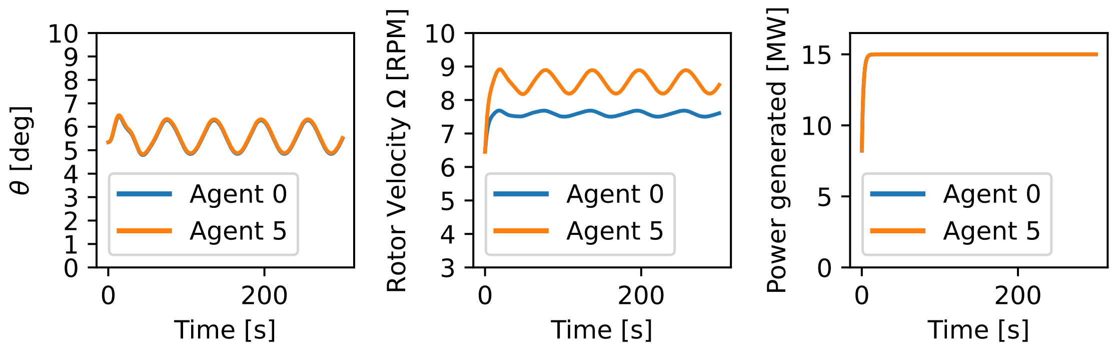


(a) Response for wind mean 16 m/s and amplitude 3 m/s, without a PSF.

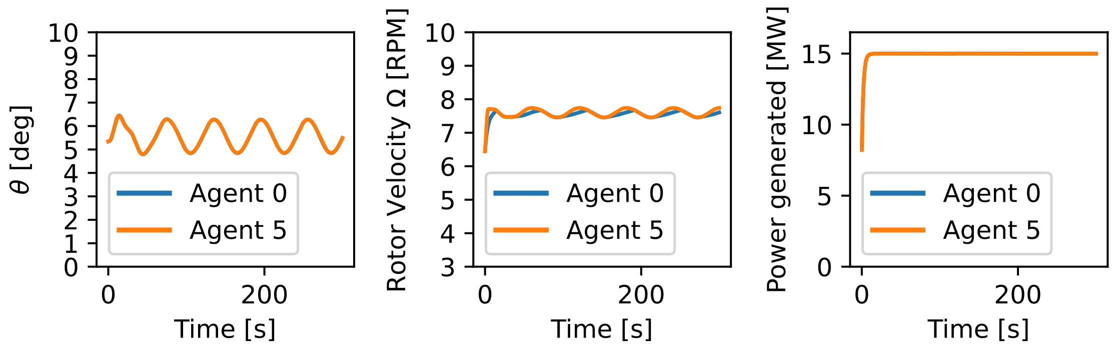


(b) Response for wind mean 16 m/s and amplitude 3 m/s, with a PSF.

Figure 4.1.14: Response of system for wind mean of 16 m/s and amplitude of 3 m/s, shown for Agent 0 and Agent 5 deployed with and without PSF.



(a) Response for wind mean 22 m/s and amplitude 3 m/s, without a PSF.



(b) Response for wind mean 22 m/s and amplitude 3 m/s, with PSF.

Figure 4.1.15: Response of system for wind mean of 22 m/s and amplitude of 3 m/s, shown for Agent 0 and Agent 5 deployed with and without filtering.

4.1.4 Main result summary

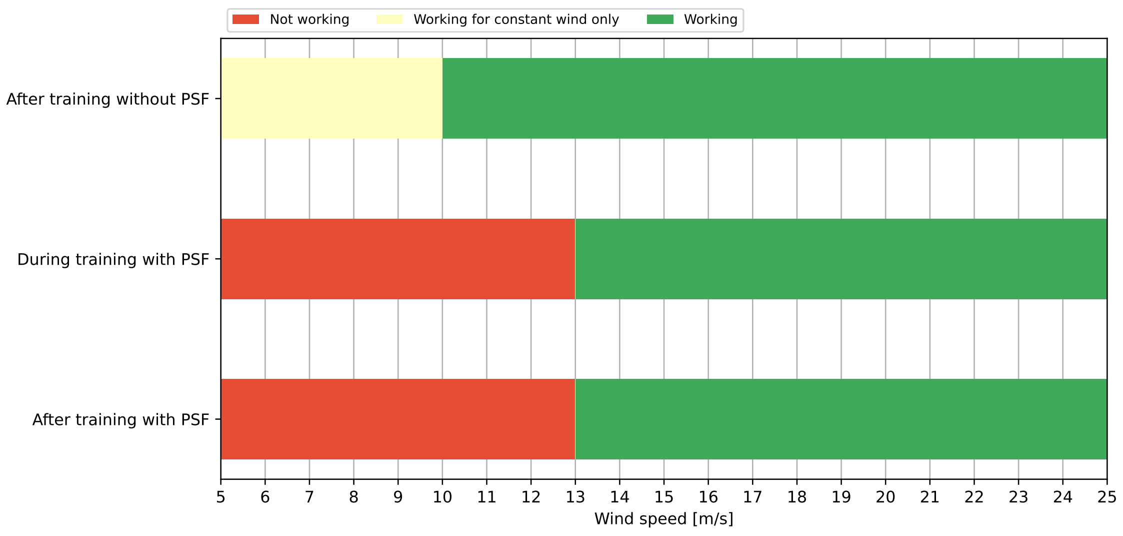


Figure 4.1.16: Overview of results expressed by degree of success at given wind speeds.

In Figure 4.1.16 we present a summary of the various cases displayed by what wind conditions they succeed in. The results show that the RL agent without the PSF is able to control the turbine to a satisfactory degree for wind speeds above 10 m/s for varying winds and all the way down to 5 m/s for constant winds. It also shows that the PSF is able to successfully prevent crashes during training when wind speeds are kept above 13 m/s but fails for lower winds. We also saw that adding the PSF accelerated learning in some cases, an example of this is shown in Figure 4.1.6. Crash prevention was preserved, and a satisfactory control strategy was created for higher winds after training when deploying the agent with filering.

4.2 Discussion

Our discussion will mainly revolve around the reasons behind the observations in section 4.1, with an emphasis on the combined RL-PSF implementation. First, we will point out some interesting observations and suggest hypotheses based on these. We will then bring forward possible sources of error and discuss their impact on the results. The errors will highlight some of our framework's shortcomings, and by doing so, also shaping its future development.

Result summary and discussion

Our results show a clear distinction between how the implementation works for low (<13 m/s) and high (>13 m/s) winds. Looking at the high wind scenarios in isolation, we see that combining the PSF with the RL has several advantages. As stated in section 1.1 and subsection 3.1.2 it is not feasible to train an RL agent on a real-world system because of safety concerns. Our implementation shows that this problem can be eliminated by adding a PSF, which we have shown to successfully eliminate crashes both during training and after deployment. This result can clearly be seen in Figure 4.1.6a. The crash prevention during training is seen as one of the most significant contributions of our work, enabling the possibility of training an RL agent on a real-world system assuming a sufficiently accurate model is present in the PSF. We have also shown that agents are able to generalize fairly well, especially when trained on higher winds only, however there is naturally some degradation in performance.

Another advantage of adding a PSF is an acceleration of the learning in certain conditions. This is exemplified in Figure 4.1.6b where the agent with filering converges much faster than the one without for Level HighWinds. This might be because the PSF restricts the agent to only exploring relevant areas of the state and action spaces, i.e. reducing the time spent exploring irrelevant areas. This becomes more apparent in Level HighWinds than in Level 0, and might be because the irrelevant areas of the state and action space becomes larger in more challenging levels.

Shifting the focus to lower winds, we see that our implementation does not perform satisfactory. An observation made regarding the agents' training progress is that, in some cases, the PSF seems to hinder the agent from learning optimal safe actions. This is clear when looking at results for Level 1-5 in Figure 4.1.3a. Figure 4.1.1a shows the agent starts out with a high crash rate but eventually learns that actions that prevent crashing are beneficial. This is not the case with filtering in Figure 4.1.3a where the crash rate is non-decreasing during the course of training. The same effect can also be seen in the connection between crash rate and reward. The crash rate and reward seem to be significantly more correlated for the agents without filtering than with the PSF.

We then investigate the difference in training with filtering by looking at the fully trained agents. As stated previously, performance and crash rate have to be considered together. So even though performance is increased for Level 2 and 5 when comparing Figure 4.1.2 and Figure 4.1.4 we have to consider that the crash rate is also increased. This suggests that the PSF causes the system to crash in the episodes that otherwise would have lead to a low performance measure. The 1% crash rate in Figure 4.1.2 showcases how there is *no guarantee* for constraint satisfaction and avoidance of negative outcomes with RL and shows why we need the PSF in real-world applications. Even if it looked like it learned not to crash during training, we could see two small peaks in the crash rate for Level 1 towards the end of training. This might be because of certain combinations of wind conditions, most likely low wind, and states that led to the challenging control problem.

Reflecting on the results observed when applying filtering to an agent trained without filtering, we do not see any significant differences in neither performance nor crash rate. Performance is not improved nor deteriorated, likewise with crash rate. In other words, the PSF still "passes through" the safe actions that lead to high performance, and still drives the system to a crash for the actions that were unsafe. This indicates that the PSF *is* able to determine if an action is safe or not but struggles to find the correct, safe action if the original one was unsafe.

Looking at the system response in subsection 4.1.3 we note a curious result; Agent 0 was able to keep the turbine from crashing for a longer time than Agent 5 in low winds. This is curious because Agent 5 is trained on lower winds, while Agent 0 is not. It indicates that the problems at low wind speeds negatively impact the agent more than not training in those conditions at all. This finding is also made in Figure 4.1.8a, where Agent 0 performs better in all levels 0-5 when compared to the agent trained in the respective level. Section 4.1.3 of course only shows one example of a response, and we need to be careful about extrapolating this to the general case, but together with Figure 4.1.8a it gives some interesting insight into the relation between model problems and agent performance.

While our focus has been mostly directed towards the response in Ω, we also want to optimize the relevant states of platform angle θ and power generation P_{gen}. Power generation is mostly kept at 15 MW by both Agent 0 and 5. However, an interesting observation is made at low wind speeds. It looks like the agents drop the power generation to prevent the rotor from slowing down too much, but this clearly happens too late to be successful. It does however point towards the expected behavior. The response in θ does not seem to be corrected much by any of the agents, both with and without filtering. A further look into F_{thr} also showed that it has a minor impact on the platform angle and that none of the agents are able to use it to suppress platform oscillations. Using the thruster to counter-act the force from the wind is useless. Compared to simulation peak wind forces at $F_{wind} = 2.6e7$, the thruster has only 0.66% of its torque. In addition, Fossen (2021, Figure 9.5) shows an example of a thrust-power curve for a variable speed fixed-pitch propeller. The figure shows that at $F_{thr} = 500$ kN, it has a power consumption of about 2750 kW, or about 18.3% of the FOWT energy production. This means that a significant amount of the generated power is going towards an attempt to keep the platform vertical that has little to no impact. Still, this kind of actuator might become more relevant in future work when adding waves, wind gusts, and other disturbances.

Increased system complexity and reward function

Diving deeper into the reasons behind the problematic results at low winds speeds, we try to find the root cause by looking at the increased system complexity. The increase in complexity of adding the PSF could be the reason to why the pure RL agent able to learn the control strategy to a high degree of success down to 10 m/s (5 m/s with constant wind), while the combined controller fails under 13 m/s. When the PSF solver struggles to find the backup trajectory, e.g., when the FOWT is close to the limits of its safe set, the possible abnormal or fluctuating outputs could misleading the RL agent. The reason for the RL agent working in constant low winds might be because the setpoints for Ω and P_{gen} (Figure 3.2.2 and 3.2.1) appear constant to the agent. We believe that the dynamical setpoints are part of the problems occurring below 13 m/s, which we discuss further later.

The reward calculation is an example of where the increased system complexity could have an impact. We acknowledge that the reward function might be over-engineered to some degree, and a simpler function could prove beneficial. As we explained in section 3.2, a number of different reward functions were explored but Equation 3.2.3 gave the best results. While changes in the reward function produce significant changes in the behavior of the agents, we believe that further development of the reward function will have the most potential for acceleration of the training and not for solving the cases where it fails to learn.

One theory on how training could be sped up is by keeping the lower limit of the reward function to zero. We see an initial dip in the episode reward mean in Figure 4.1.1b. This might be because the reward is negative at each timestep (due to bad general performance), giving a lower cumulative reward the longer the agent survives. This means that the agent needs to increase the performance by figuring out the other reward terms, such that the reward at each timestep is above zero before it can learn that the survival is beneficial. Keeping the reward function strictly positive might avoid this problem and possibly also accelerate training. A side note here is that, as explained in section 3.2, we want to avoid a reward function gradient of zero far from the optimum, as this might make it hard to optimize by gradient descent. Note that making a reward function for RL is highly dependent on the specific application and we point out that we do not claim to have found the optimal reward function in any way. Finding the optimal reward function for any problem is seen as a near-impossible task, and there is usually room for improvement.

Predictive safety filter limitations

The overall results regarding the PSF from subsection 4.1.1 show that as soon as wind speeds below 13 m/s are included in the training. The PSF struggles to find correct solutions resulting in more crashes than without the filtering, even for constant wind. There might be multiple reasons for this.

One of the more obvious reasons why the PSF fails during low variable winds could be the assumption of constant wind made in the PSF. The inherent robustness of nominal MPC is the key assumption for being able to move to slow varying wind with a nominal PSF formulation. The robustness properties does perhaps not transfer to the PSF formulation to the sufficient degree. In contrast to MPC, the PSF does not drive the state trajectory toward a reference and could result in a trajectory closer to the "stability boarder" of this system.

Nevertheless, this does not explain why it also fails to succeed at low *constant* winds. Another simplification made in the PSF is the exclusion of the model for adjusted wind (Equation 3.1.6). The PSF bypasses this by using the adjusted wind speed w directly without including the analytical model for it. This means that even though w_0 is constant, the wind sent to the PSF w might still vary, possibly producing problems with regards to the constant wind assumption.

Recall that to speed up the training process, we created an alternate terminal set, and in doing so, we break the recursive feasibility required to make a claim on its provable stability. Some crashing was expected due to this simplification. However, to confirm its impact, we redid some of our experiments with both the true robust ellipsoidal set and the steady-state terminal set discussed in subsection 3.2.4. This was done in both constant and variable wind, and the outcome did not change significantly when using our alternate set. Had the alternate terminal set been the sole problem, we would expect this to eliminate most of the crashes for constant wind. This was not the case.

Difficulties might also be a result of the tighter constraints being added to the PSF. As exemplified by Figure 3.2.5, it was observed that the RL agent was not able to adapt to the tighter constraints. We acknowledge that, given the poor performance, the tight constraints are hard to justify. However, we note the justification given in subsection 3.2.5, and further add that we still observed similar problems with the PSF when relaxing our constraints.

The fact that the PSF successfully prevents crashes during training for levels only containing winds above 13 m/s, including varying winds, suggests that it is not the variation in the wind that is the main problem for the PSF but rather low wind speeds. An interesting thing to note is that the RL agent without PSF also struggles for low winds, especially when winds below 10 m/s are included. This suggests that the problem could be with the model itself and not necessarily the PSF.

Model difficulties and numerical errors

While problems related to our model's invalidity are probable, another theory is that wind turbines are inherently difficult to control (Butterfield et al., 2007). This was also observed during experimentation with the high-fidelity model in OpenFAST. There were cases where the states exploded, and the turbine became unstable. Even with the inherent difficulties of turbine control, we believe it is probable that our significantly simplified model is the main root of our problems.

As stated early in the thesis, our model is meant as a coarse approximation to a FOWT and contains several simplifications and assumptions that might make it inaccurate. The model simplification with regards to adjusted wind speed in Equation 3.1.6, e.g., static inflow, could be a possible source of error. However, investigating the platform's sway velocity $\dot{\theta}$, we have observed that this amounts to changes less than 1 m/s. While contributing to possible failures, we claim that this simplification should not cause the amount of crashes that is reported.

As expressed in subsection 3.2.5, we observed an increase in solver convergence when changing the cost function of the PSF formulation. This points towards one of the major issues being with our model formulation. We expressed our model using SI units, leaving a huge difference in the input range. While we scaled our cost function to alleviate some numerical errors, we did not reformulate the model itself, as done in the terminal set optimization. Inspecting the feedback system $\mathbf{A}(\boldsymbol{\delta}) - \mathbf{B}(\boldsymbol{\delta})\mathbf{K}$, we can observe to some degree the impact of the normalization. Given that eigenvalues are in the left half-plane, the real part dictates the rate of decay in the transient solution, which in turn is associated with permissible discretization timestep τ. With the scaling of $\mathbf{B}(\boldsymbol{\delta})$ discussed in subsection 3.2.4, we find the largest eigenvalue $\overline{\lambda_{norm}}$ of the feedback system to be $\Re(\overline{\lambda_{norm}}) \approx -27$. Without the normalization we found the largest eigenvalue $\overline{\lambda}$ to be $\Re(\overline{\lambda}) \approx -6000$. The normalization clearly impacts the permissible timestep, but the values mostly have to be considered in relation to each other as the linear system approximates the nonlinear FOWT model. Using the stability function $R_E(\mu)$ in Equation 2.1.14, we find that

$$|R_E(\tau * \overline{\lambda_{norm}})| = |R_E(0.1 * \overline{\lambda_{norm}}) \approx 0.99 \tag{4.2.1}$$

$$|R_E(\tau * \overline{\lambda})| = |R_E(0.1 * \overline{\lambda})| \approx 5.36e9 \tag{4.2.2}$$

The non-normalized linear feedback system has poor stability properties with regard to discretization. Disregarding the input matrix $\mathbf{B}(\boldsymbol{\delta})$, we still observe poor discretization stability for the system matrix $\mathbf{A}(\boldsymbol{\delta})$. The low stability is expressed as values close to the stability borders for a wide array of step sizes for RK4, and as result, decreasing the stepsize does not necessary solve the discretization problem. Without drawing solid conclusions between the linear system and the actual model, we would argue that it makes a strong case for the results we are seeing. This also extends to the RL agent acting on the environment alone, where this could manifest itself as the reason why the agents performs poorly for low winds.

Having said that, we see that the numerical issues do not impact all of our experiments to the same degree. Looking at the rotor thrust curve in Figure 4.2.1, we see that, perhaps counter-intuitively, the axial thrust is peaking at 10.59 m/s. At the same time, our desired power curve is going from static to dynamic once we cross the same point. Thus, the increased complexity of dynamic setpoints is believed to act as a catalyst for instability in our mathematical model, leaving the model close to uncontrollable at lower wind speeds. A constant environment wind speed does not necessarily imply a constant adjusted wind speed, as the platform movement induces varying wind. We use the term *close to* and don't claim complete uncontrollability because we are able to find a steady-state solution to initialize the environment for all the considered wind speeds (5-25 m/s). This means our model can at least be steady-state stabilized for each wind speed without placing any claims on the transients or varying wind cases.

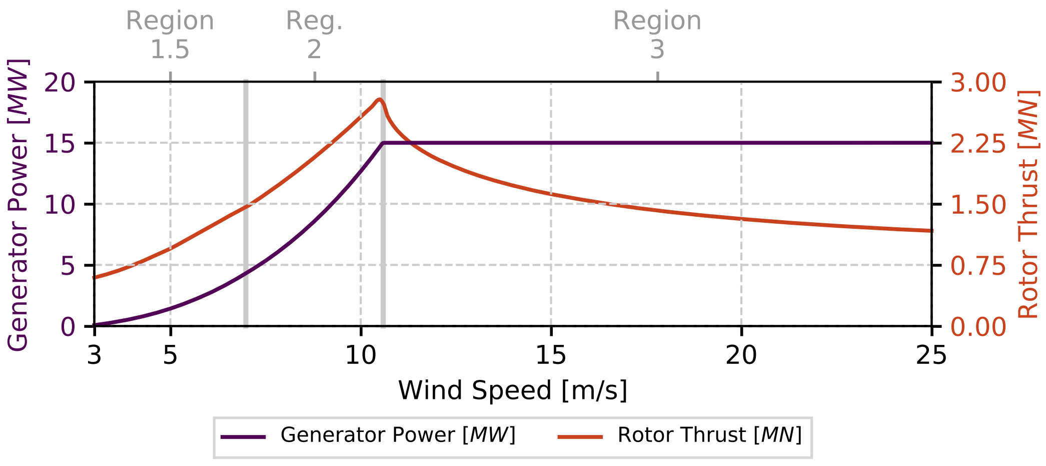


Figure 4.2.1: Thrust power curve from Gaertner et al. (2020).

Summary

Through this discussion we have highlighted the contribution of our work with a successful implementation under certain conditions. However, we have also brought attention to multiple limitations in our work and probable causes for the reported problems. In summary, we conclude that the combination of the model difficulties, namely low numerical stability and a complex system with dynamical setpoints, in conjunction with our PSF simplifications are believed to be the main contributors. For higher wind speeds where the model is well behaved and the setpoints are constant, we produce a successful implementation where RL and the PSF are working together to learn a close to optimal control policy with crash prevention both during training and deployment.

Chapter 5
Conclusion and future work

In this chapter, we conclude the report, reflect on our work, and outline some alternatives for future work within this application.

5.1 Conclusions

Within the scope of the project a reinforcement learning (RL) with predictive safety filtering (PSF) framework was developed and tested using the application of a floating offshore wind turbine (FOWT). The major conclusions of the thesis can be itemized as follows:

- We have shown that an RL agent is able to control our model of the FOWT for slowly varying winds with fluctuations of up to 3 m/s as long as the wind speed stays above 10 m/s. For wind speeds below this, the agent fails to produce a satisfactory control policy. The reasons behind this were discussed, with modeling issues regarding numerical instability being presented as the most probable.
- The PSF is able to prevent crashes both during the training and after the deployment of the RL agent for slowly varying winds with fluctuations of up to 3 m/s as long as the wind speed is kept above 13 m/s, but struggles as soon as lower wind speeds are encountered. Again, modeling issues along with the PSF simplifications are presented as the most probable causes.
- The RL agent's ability to find an adequate control optimum and learn the system dynamics was not degraded by adding PSF. In fact, adding the PSF accelerated the training under high wind conditions.

In doing so, we answer the research questions mentioned in subsection 1.2.2, thereby realizing all the project objectives. The framework's high level of modularity is beneficial for future research and will hopefully facilitate further development.

5.2 Writer reflection

While we believe our work to be a good starting point for investigating this direction within control, several things could have been done differently in hindsight. The first thing that comes to mind is that more investigation into the model validity and accuracy in different wind conditions. This should have been performed before further implementation in varying wind. Secondly, we should have made a bolder move, and moved directly to learning-based PSF. Our reasoning behind not doing so, was to first be able to use the simpler case of nominal PSF formulation, before moving on to significantly more complex controller. With the new paper Wabersich and Zeilinger (2021*b*) coming out during our work this approach only got more relevant. However, this was discovered too late with the paper not becoming available online until April. Because our framework is highly modular, it is seen as a great opportunity for future research to investigate incorporating this learning-based approach. More detail on this in section 5.3. With all of that in mind, our work was meant to investigate the use of PSF and RL together and we believe that this was done successfully, referring to the results for high winds.

5.3 Recommendations for future work

5.3.1 Model improvement

The models used in this report are based on a number of assumptions and approximations. This is reflected by the issues discussed in section 4.2. Further work should investigate incorporating more complex and accurate models in both the model used by the RL and the approximated model in the PSF.

One direction to go is extending the current model by eliminating some of our assumptions and make it more accurate. Examples are adding a more complex and accurate inflow model, investigating the problems with low stability margins for the current system, or making the platform model more realistic, e.g., by removing the assumption of a fixed rotation point.

Another direction could be to use a neural network to represent the input-output dynamics of the system. This was done successfully in Zhang et al. (2019), where a neural network was trained on a dataset created by the high-fidelity model in OpenFAST. As mentioned previously, this could replace the *plant* block in Figure 3.1.1 resulting in a more accurate representation of the real dynamics. This way of modeling is also extremely relevant when working with RL, as one of the main advantages of RL is not having a need for an explicit analytical model. Continuing in the data driven modeling direction, a natural extension to the wind model could be to use physics guided machine learning (PGML) to calculate the forces and torques more accurately and efficiently from the full wind field. The PGML approach makes incorporation of domain specific knowledge into a neural network model feasible.

Adding more realistic platform actuation

As discussed in section 4.2, a thruster might not be the most realistic form of actuation for the platform. This is especially true when looking at the ratio between power consumption and impact on the system. At full thrust the platform thruster uses about 18.3% of the generated power and still shows a low impact on the response of the turbine.

5.3.2 Alternative reinforcement learning algorithms and hyperparameter tuning

This report uses an implementation with PPO as the RL algorithm, and is due to the findings in Teigen (2020). As metioned before, it was found that PPO was significantly better than other algorithms for a problem with similar state and action spaces. However, the choice of algorithm is highly application dependent and other algorithms like SAC (Haarnoja et al., 2018), DDPG (Lillicrap et al., 2015) or TD3 (Fujimoto et al., 2018) might perform better in this specific application. The hyperparameters of an algorithm can have a major impact on training performance. This is well known by the community, and was also observed for this project. Further tuning of hyperparameters could accelerate learning even further.

There exists a number of frameworks for automatic hyperparameter tuning, including the one mentioned by Stable Baselines3 docs, Optuna (Akiba et al., 2019). An alternative is replacing Stable Baselines3 with something like Ray RLlib which includes several extremely powerful tools, one of which is Ray Tune for hyperparameter tuning. Other advantages are Microsoft Azure integration, high scalability as well as multi agent support.

5.3.3 Predictive safety filter module

As we have seen, the FOWT model used throughout this thesis has several limitations both on the numerical side and in its simplification of the dynamics. From a control perspective, the discrepancies between the model and the actual physical system has to be sufficiently small for stability proofs to be applicable. Improving the accuracy of the model is therefore also paramount for the applicability and validity of the PSF induced stability. However, the physical system will be ever-changing and motivates the use of adaptive and learning-based methods. This is in our opinion the most natural extension of our work. As mentioned in subsection 1.1.1, the PSF formulation in Wabersich and Zeilinger (2018*b*), and more recently Wabersich and Zeilinger (2021*b*), has two key features not implemented in our framework; (1) The PSF is learning-based, and (2) it uses robust MPC principles for disturbance rejection. This continuation could eliminate many of the PSF limitations addressed in section 4.2. In addition, this learning-based approach makes more sense in combination with the model-free RL as it will remove the need for a model all together.

Other approaches to retaining the model but addressing assumption of sufficient robustness of the PSF, are extensions leading to provable robustness. Tube-based nonlinear MPC formulations (Mayne et al., 2011) could be adapted within the framework to reclaim the provable stability of the PSF formulation under varying wind conditions. Another extension could be to introduce a scenario-based MPC as depicted in Schildbach et al. (2014). While the complexity of the system could increase, reclaiming the provable stability the framework could offer is of great importance.

Bibliography

Abdullah, M., Yatim, A., Tan, C. and Saidur, R. (2012), 'A review of maximum power point tracking algorithms for wind energy systems', *Renewable and Sustainable Energy Reviews* **16**(5), 3220–3227.
URL: *https://www.sciencedirect.com/science/article/pii/S1364032112001098*

Agrawal, A., Diamond, S. and Boyd, S. (2019), 'Disciplined geometric programming', *Optimization Letters* **13**(5), 961–976.

Agrawal, A., Verschueren, R., Diamond, S. and Boyd, S. (2018), 'A rewriting system for convex optimization problems', *Journal of Control and Decision* **5**(1), 42–60.

Ajay Menon (2021), 'What is bollard pull – everything you wanted to know'. [Online; Last accessed 14-May-2021].
URL: *https://www.marineinsight.com/naval-architecture/bollard-pull-everything-you-wanted-to-know/*

Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M. (2019), Optuna: A next-generation hyperparameter optimization framework, *in* 'Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining'.

Allen, C., Viselli, A., Dagher, H., Goupee, A., Gaertner, E., Abbas, N., Hall, M. and Barter, G. (2020), 'Definition of the UMaine VolturnUS-S reference platform developed for the IEA Wind 15-megawatt offshore reference wind turbine', NREL/TP-76773.

Allgöwer, F. and Zheng, A. (2012), *Nonlinear model predictive control*, Vol. 26, Birkhäuser.

Ames, A. D., Grizzle, J. W. and Tabuada, P. (2014), Control barrier function based quadratic programs with application to adaptive cruise control, *in* '53rd IEEE Conference on Decision and Control', IEEE, pp. 6271–6278.

Ames, A. D., Xu, X., Grizzle, J. W. and Tabuada, P. (2016), 'Control barrier function based quadratic programs for safety critical systems', *IEEE Transactions on Automatic Control* **62**(8), 3861–3876.

Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B. and Diehl, M. (2019), ‘Casadi: a software framework for nonlinear optimization and optimal control’, *Mathematical Programming Computation* **11**(1), 1–36.

ApS, M. (2019), ‘Mosek optimization toolbox for matlab’, *User’s Guide and Reference Manual, version* **4**.

Bemporad, A. and Morari, M. (1999), Robust model predictive control: A survey, *in* ‘Robustness in identification and control’, Springer, pp. 207–226.

Biegler, L. T. and Zavala, V. M. (2009), ‘Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization’, *Computers & Chemical Engineering* **33**(3), 575–582.

Boyd, S., El Ghaoui, L., Feron, E. and Balakrishnan, V. (1994), *Linear matrix inequalities in system and control theory*, SIAM.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016), ‘Openai gym’.

Butterfield, S., Musial, W., Jonkman, J. and Sclavounos, P. (2007), ‘Engineering challenges for floating offshore wind turbines’.
URL: *https://www.osti.gov/biblio/917212*

Choi, J., Castaneda, F., Tomlin, C. J. and Sreenath, K. (2020), ‘Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions’, *arXiv preprint arXiv:2004.07584* .

Diamond, S. and Boyd, S. (2016), ‘CVXPY: A Python-embedded modeling language for convex optimization’, *Journal of Machine Learning Research* **17**(83), 1–5.

Egeland, O. and Gravdahl, J. T. (2002), *Modeling and simulation for automatic control*, Vol. 76, Marine Cybernetics Trondheim, Norway.

Europe, W. (2019), ‘Offshore wind bin europe, key trends and statistics 2019’.

Fehlberg, E. (1970), ‘Klassische runge-kutta-formeln vierter und niedrigerer ordnung mit schrittweiten-kontrolle und ihre anwendung auf waermeleitungsprobleme’, *Computing* **6**(1-2), 61–71.

Fossen, T. (2021), *Handbook of Marine Craft Hydrodynamics and Motion Control*, John Wiley & Sons.

Fujimoto, S., van Hoof, H. and Meger, D. (2018), ‘Addressing function approximation error in actor-critic methods’.

Gaertner, E., Rinker, J., Sethuraman, L., Zahle, F., Anderson, B., Barter, G., Abbas, N., Meng, F., Bortolotti, P., Skrzypinski, W., Scott, G., Feil, R., Bredmose, H., Dykes, K., Sheilds, M., Allen, C. and Viselli, A. (2020), 'Definition of the IEA 15-megawatt offshore reference wind turbine', NREL/TP-75698.
URL: *https://www.nrel.gov/docs/fy20osti/75698.pdf*

Garcia, J. and Fernández, F. (2012), 'Safe exploration of state and action spaces in reinforcement learning', *Journal of Artificial Intelligence Research* **45**, 515–564.

García, J. and Fernández, F. (2015), 'A comprehensive survey on safe reinforcement learning', **16**, 1437–1480.

Gaskett, C. (2003), 'Reinforcement learning under circumstances beyond its control'.

Gehring, C. and Precup, D. (2013), Smart exploration in reinforcement learning using absolute temporal difference errors, *in* 'Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems', pp. 1037–1044.

Geibel, P. and Wysotzki, F. (2005), 'Risk-sensitive reinforcement learning applied to control under constraints', *Journal of Artificial Intelligence Research* **24**, 81–108.

Global Wind Energy Council, G. (2021), 'Global wind report 2021', `https://gwec.net/wp-content/uploads/2021/03/GWEC-Global-Wind-Report-2021.pdf`.

Gros, S. and Schild, A. (2017), 'Real-time economic nonlinear model predictive control for wind turbine control', *International Journal of Control* **90**(12), 2799–2812.

Grüne, L. (2012), 'Nmpc without terminal constraints', *IFAC Proceedings Volumes* **45**(17), 1–13.

Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S. (2018), 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor'.

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L. and Ostendorf, M. (2016), 'Deep reinforcement learning with a natural language action space'.

Hewing, L., Wabersich, K. P., Menner, M. and Zeilinger, M. N. (2020), 'Learning-based model predictive control: Toward safe learning in control', *Annual Review of Control, Robotics, and Autonomous Systems* **3**, 269–296.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S. and Wu, Y. (2018), 'Stable baselines', `https://github.com/hill-a/stable-baselines`.

IEA (2020), 'Offshore wind power generation in the sustainable development scenario, 2000-2030', https://www.iea.org/data-and-statistics/charts/offshore-wind-power-generation-in-the-sustainable-development-scenario-2000-2030.

IPCC (2021), 'The intergovernmental panel on climate change, fns klimapanel', `https://www.ipcc.ch/`.

Jafarnejadsani, H., Pieper, J. and Ehlers, J. (2012), Adaptive control of a variable-speed variable-pitch wind turbine using rbf neural network, *in* '2012 IEEE Electrical Power and Energy Conference', pp. 216–222.

JM, J., Butterfield, S., Musial, W. and Scott, G. (2009), 'Definition of a 5mw reference wind turbine for offshore system development', *National Renewable Energy Laboratory (NREL)* .

Johansen, T. A. (2011), 'Introduction to nonlinear model predictive control and moving horizon estimation', *Selected topics on constrained and nonlinear control* **1**, 1–53.

Jonkman, J., Buhl, M., Hayman, G., Jonkman, B., Mudafort, R., Platt, A. and Sprague, M. (2021), 'Openfast documentation'.
URL: *https://openfast.readthedocs.io/en/main/*

Kamthe, S. and Deisenroth, M. (2018), Data-efficient reinforcement learning with probabilistic model predictive control, *in* 'International Conference on Artificial Intelligence and Statistics', PMLR, pp. 1701–1710.

Khalil, H. K. (2015), *Nonlinear systems*, Vol. 3, Prentice Hall.

Kumar, A. and Stol, K. (2009), Scheduled model predictive control of a wind turbine, *in* '47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition', p. 481.

Leineweber, D. B., Bauer, I., Bock, H. G. and Schlöder, J. P. (2003), 'An efficient multiple shooting based reduced sqp strategy for large-scale dynamic process optimization. part 1: theoretical aspects', *Computers & Chemical Engineering* **27**(2), 157–166.

Li, Y. (2018), 'Deep reinforcement learning: An overview'.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. (2015), 'Continuous control with deep reinforcement learning'.

Löfberg, J. (2004), Yalmip : A toolbox for modeling and optimization in matlab, *in* 'In Proceedings of the CACSD Conference', Taipei, Taiwan.

Magni, L., Raimondo, D. M. and Allgöwer, F. (2009), 'Nonlinear model predictive control', *Lecture Notes in Control and Information Sciences* **384**.

Mayne, D. Q. (2014), 'Model predictive control: Recent developments and future promise', *Automatica* **50**(12), 2967–2986.

Mayne, D. Q., Kerrigan, E. C., Van Wyk, E. and Falugi, P. (2011), 'Tube-based robust nonlinear model predictive control', *International Journal of Robust and Nonlinear Control* **21**(11), 1341–1353.

Mayne, D. Q., Rawlings, J. B., Rao, C. V. and Scokaert, P. O. (2000), 'Constrained model predictive control: Stability and optimality', *Automatica* **36**(6), 789–814.

Mehrizi-Sani, A. (2017), Chapter 2 - distributed control techniques in microgrids, *in* M. S. Mahmoud, ed., 'Microgrid', Butterworth-Heinemann, pp. 43 – 62.
URL: *https://bit.ly/3hrWSMI4*

Meyer, E. (2020), 'On course towards model-free guidance, a self-learning approach to dynamic collision avoidance for autonomous surface vehicles'.

Niroui, F., Zhang, K., Kashino, Z. and Nejat, G. (2019), 'Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments'.

NTB (2021), 'Empire wind 2 and beacon wind', `https://www.tu.no/artikler/equinor-vant-kjempekontrakt-for-havvind-i-new-york/505513`. Accessed: 2021-03-19.

Paden, B., Čáp, M., Yong, S. Z., Yershov, D. and Frazzoli, E. (2016), 'A survey of motion planning and control techniques for self-driving urban vehicles', *IEEE Transactions on intelligent vehicles* **1**(1), 33–55.

Panigrahy, R. (2004), 'Minimum enclosing polytope in high dimensions', *arXiv preprint cs/0407020* .

Pedersen, M. D. (2017), 'Stabilization of floating wind turbines'.

Qin, S. J. and Badgwell, T. A. (1997), An overview of industrial model predictive control technology, *in* 'AIche symposium series', Vol. 93, New York, NY: American Institute of Chemical Engineers, 1971-c2002., pp. 232–256.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A. and Dormann, N. (2019), 'Stable baselines3', `https://github.com/DLR-RM/stable-baselines3`.

Robey, A., Hu, H., Lindemann, L., Zhang, H., Dimarogonas, D. V., Tu, S. and Matni, N. (2020), Learning control barrier functions from expert demonstrations, *in* '2020 59th IEEE Conference on Decision and Control (CDC)', IEEE, pp. 3717–3724.

Sato, M., Kimura, H. and Kobayashi, S. (2001), 'Td algorithm for the variance of return and mean-variance reinforcement learning', *Transactions of the Japanese Society for Artificial Intelligence* **16**(3), 353–362.

Scherer, C. and Weiland, S. (2000), 'Linear matrix inequalities in control', *Lecture Notes, Dutch Institute for Systems and Control, Delft, The Netherlands* **3**(2).

Schildbach, G., Fagiano, L., Frei, C. and Morari, M. (2014), 'The scenario approach for stochastic model predictive control with bounds on closed-loop constraint violations', *Automatica* **50**(12), 3009–3018.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I. and Abbeel, P. (2017), 'Trust region policy optimization'.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017), 'Proximal policy optimization algorithms'.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016), 'Mastering the game of go with deep neural networks and tree search.'.

Själander, M., Jahre, M., Tufte, G. and Reissmann, N. (2019), 'EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure', https://www.hpc.ntnu.no/idun.

Sutton, R. S. and Barto, A. G. (2018), *Reinforcement Learning: An Introduction, second ed.*, The MIT Press.

Taha, H. A. (2013), *Operations research: an introduction*, Pearson Education India.

Teigen, H. Ø. (2020), 'Investigating performance of deep reinforcement learning algorithms for path-following and collision avoidance in autonomous vessels'.

Teigen, H. Ø. and Malmin, V. (2021), 'RL-PSF github repository', https://github.com/TTK4900-RL-PSF/RL-PSF.

Van der Hoven, I. (1957), 'Power spectrum of horizontal wind speed in the frequency range from 0.0007 to 900 cycles per hour', *Journal of Atmospheric Sciences* **14**(2), 160 – 164.
URL: *"https://journals.ametsoc.org/view/journals/atsc/14/2/1520-0469_1957_014_0160_psohws_2_0_co_2.xml"*

Vandenberghe, L. (2010), 'The cvxopt linear and quadratic cone program solvers', *Online: http://cvxopt. org/documentation/coneprog. pdf* .

Vineyard Wind 1 (2021), https://www.vineyardwind.com/vineyard-wind-1. Accessed: 13.01.2021.

Von Stryk, O. (1993), Numerical solution of optimal control problems by direct collocation, *in* 'Optimal control', Springer, pp. 129–143.

Wabersich, K. P. and Zeilinger, M. N. (2018*a*), Linear model predictive safety certification for learning-based control, *in* '2018 IEEE Conference on Decision and Control (CDC)', IEEE, pp. 7130–7135.

Wabersich, K. P. and Zeilinger, M. N. (2018*b*), 'Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning', *arXiv preprint arXiv:1812.05506* .

Wabersich, K. P. and Zeilinger, M. N. (2021*a*), 'A predictive safety filter for learning-based control of constrained nonlinear dynamical systems'.

Wabersich, K. P. and Zeilinger, M. N. (2021*b*), 'A predictive safety filter for learning-based control of constrained nonlinear dynamical systems', *Automatica* **129**, 109597.

Wang, L. (2001), 'Continuous time model predictive control design using orthonormal functions', *International Journal of Control* **74**(16), 1588–1600.

Wang, L., Han, D. and Egerstedt, M. (2018), Permissive barrier certificates for safe stabilization using sum-of-squares, *in* '2018 Annual American Control Conference (ACC)', IEEE, pp. 585–590.

Wieland, P. and Allgöwer, F. (2007), 'Constructive safety using control barrier functions', *IFAC Proceedings Volumes* **40**(12), 462–467.

World Resources Institute, W. (2021), 'Climate watch historical country greenhouse gas emissions data (1990-2018)', `https://www.climatewatchdata.org/ghg-emissions?breakBy=sector&chartType=percentage&end_year=2018&start_year=1990`. Accessed: 2021-03-19.

Wu, Y., Mansimov, E., Liao, S., Radford, A. and Schulman, J. (2017), 'Openai baselines: Acktr & a2c'.

Xie, H., Xu, X., Li, Y., Hong, W. and Shi, J. (2020), Model predictive control guided reinforcement learning control scheme, *in* '2020 International Joint Conference on Neural Networks (IJCNN)', IEEE, pp. 1–8.

Zhang, F. (2006), *The Schur complement and its applications*, Vol. 4, Springer Science & Business Media.

Zhang, J., Zhao, X. and Wei, X. (2019), 'Reinforcement learning-based structural control of floating wind turbines'.

Zhang, K. and Luo, M. (2015), 'Outlier-robust extreme learning machine for regression problems', *Neurocomputing* **151**, 1519–1527.

Appendix A

Algorithms

Algorithm 1: Terminal set

```
Data:
System dynamics $\dot{\mathbf{x}}$,
Polytopic state constraint $\mathcal{X}$,
Polytopic input constraint $\mathcal{U}$,
Polytopic external parameter constraint $\mathcal{P}$.
Result:
Ellipsoidal terminal set matrix $\mathbf{P}$,
Ellipsoidal center $\mathbf{x}_{c_0}$.
begin
    /* Given by Equation 2.1.3                                  */
    $\mathbf{A}, \mathbf{B} \leftarrow nonlinearToLinear(\dot{\mathbf{x}})$
    /* Given by Equation 3.1.13                                 */
    $\mathcal{A}, \mathcal{B} \leftarrow polytopicUncertaintySet(\mathbf{A}, \mathbf{B}, \mathcal{X}, \mathcal{U}, \mathcal{P})$
    /* Given by Equation 3.1.14                                 */
    $\mathbf{x}_{c_0}, \mathbf{u}_{c_0} \leftarrow centerOptimization(\dot{\mathbf{x}}, \mathcal{X}_{\mathcal{T}}, \mathcal{U}_{\mathcal{T}})$
    /* Given by Equation 2.1.6 and Equation 2.1.11              */
    $\mathcal{A}_c, \mathcal{B}_c, \mathcal{X}_c, \mathcal{U}_c \leftarrow moveSystem(\mathcal{A}, \mathcal{B}, \mathcal{X}, \mathcal{U}, \mathbf{x}_{c_0}, \mathbf{u}_{c_0})$
    /* Described in subsection 3.2.4                            */
    $\mathcal{B}_c, \mathcal{U}_c \leftarrow rowScaling(\mathcal{B}, \mathcal{U})$
    $\mathcal{U}_c \leftarrow columnScaling(\mathcal{U})$
    /* Described in subsection 2.4.1                            */
    $\mathbf{P} \leftarrow robustEllipsoid(\mathcal{A}_c, \mathcal{B}_c, \mathcal{X}, \mathcal{U}, \mathbf{x}_{c_0}, \mathbf{u}_{c_0})$
end
```

Malmin & Ødegård Teigen

RL and PSF for Safe AI in FOWT Control