OPTIMIZED WIND TURBINE PERFORMANCE VIA AI-BASED COLLECTIVE PITCH ANGLE CONTROL

By

Abdelhamid Nabeel Sadek Rizk Younes

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electric Power Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2024

OPTIMIZED WIND TURBINE PERFORMANCE VIA AI-BASED COLLECTIVE PITCH ANGLE CONTROL

By
Abdelhamid Nabeel Sadek Rizk Younes

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electric Power Engineering

Under the Supervision of

Prof. Dr. Essam El-Din Mohamed Aboul Zahab

.....

Professor of
Electric Power Department
Faculty of Engineering, Cairo University

Prof. Dr. Abdel-Latif Mohamed Rajaei El-Shafei

.....

Professor
Electric Power Department
Faculty of Engineering, Cairo University

Assoc. Prof. Dr. Ahmed Abdel Nasser Ahmed Lasheen

.....

Associate Professor
Electric Power Department
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2024

OPTIMIZED WIND TURBINE PERFORMANCE VIA AI-BASED COLLECTIVE PITCH ANGLE CONTROL

By
Abdelhamid Nabeel Sadek Rizk Younes

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electric Power Engineering

Approved by the
Examining Committee

_____
Prof. Dr. Essam El-Din Aboul Zahab Thesis Main Advisor

_____
Prof. Dr. Abdel-Latif Rajaei El-Shafei Advisor

_____
Assoc. Prof. Dr. Ahmed Abdel Nasser Lasheen Advisor

_____
Prof. Dr. Khaled Ali El-Metwally Internal Examiner

_____
Prof. Dr. Hany Mohamed Hasanin Mohamed External Examiner
- (Professor at Faculty of Engineering Ain Shams University)

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2024

Engineer's Name: Abdelhamid Nabeel Sadek Rizk Younes
Date of Birth: 16 / 12 / 1994
Nationality: Egyptian
E-mail: Abdelhamid.Nabeel@eng.cu.edu.eg
Phone: +201092125925
Address: 15 May city extension, Cairo, Egypt

Registration Date: 1 / 3 / 2020
Awarding Date: / / 2024
Degree: Master of Science
Department: Electric Power Engineering

Supervisors:

Prof. Essam El-Din Mohamed Aboul Zahab
Prof. Abdel-Latif Mohamed Rajaei El-Shafei
Assoc. Prof. Ahmed Abdel Nasser Lasheen

Examiners:

Prof. Essam El-Din Aboul Zahab (Thesis main advisor)
Prof. Abdel-Latif Rajaei El-Shafei (advisor)
Assoc. Prof. Ahmed Abdel Nasser Lasheen (advisor)
Prof. Khaled Ali El-Metwally (Internal examiner)
Prof. Hany Mohamed Hasanin (External examiner)

- (Professor at Faculty of Engineering Ain Shams University)

Title of Thesis:

Optimized Wind Turbine Performance via AI-based Collective Pitch Angle Control

Key Words:
Pitch angle control; Wind turbines; Reinforcement Learning; Fuzzy Logic; Neural Networks

Summary:

The thesis presents the development and evaluative study of two novel model-free artificial intelligence (AI) based collective pitch control strategies for wind energy conversion systems (WECS). It aims to achieve stabilization of generator speed and power output at rated values, diminish fluctuations, lower computational expenses, and improve overall system optimality. The initial approach introduces a cascaded-forward neural network-based controller (CNN-C), utilizing a supervised learning paradigm for its configuration. The subsequent strategy proposes a fuzzy-based deep deterministic policy gradient (F-DDPG) controller, incorporating imitation learning and deep reinforcement learning in its design. The simulation tests are conducted using the high-fidelity Wind Energy Conversion Systems (WECS) OpenFAST/MATLAB/Simulink simulation tools.

Disclaimer

I hereby declare that this thesis is my own original work and that no part of it has been submitted for a degree qualification at any other university or institute.

I further declare that I have appropriately acknowledged all sources used and have cited them in the references section.

Name: Abdelhamid Nabeel Sadek Rizk Younes Date: / /2024
Signature:

Acknowledgments

First and foremost, I would like to thank and praise Allah for all his blessings and for guidance in my life and this thesis. I would like to extend my sincerest gratitude to my thesis advisors, Prof. Essam El-Din Aboul Zahab, Prof. Abdel-Latif Rajaei El-Shafei, and Associate Prof. Ahmed Abdel Nasser Lasheen for their invaluable guidance, unwavering support, and intellectual mentorship throughout the course of this research. Your wisdom and dedication have been a beacon of light as I navigated through the complexities of this thesis. I must express my profound gratitude to my family, for their endless love, emotional support, and unwavering belief in me. Your sacrifices and resilience have not gone unnoticed, and this achievement is as much yours as it is mine. In summary, I am thankful for every individual who has been a part of this research journey, either directly or indirectly. Any accomplishments this thesis signifies are a collective triumph of all who have supported me.

List of Acronyms

ADAGRAD	Adaptive Gradient Optimizer
ADAM	Adaptive Moment Estimation
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Networks
BELBIC	Brain Emotional Learning-Based Intelligent Controller
CNN	Cascaded-forward Neural Network
CNN-C	Cascaded-forward Neural Network-based Controller
CPC	Collective Pitch Control
DDPG	Deep Deterministic Policy Gradient
DE	Differential Evolution
DELM	Deep Extreme Learning Machine
DOFs	Degrees of Freedom
DP	Dynamic Programming
DQN	Deep Quality-Networks
ELM	Extreme Learning Machine
F-DDPG	Fuzzy-based Deep Deterministic Policy Gradient
FFBP-NN	Feed Forward Back Propagation Neural Network
FMPC	Fuzzy-based Model Predictive Controller
GP_LLJ	Great Plains Low-Level Jet
GS-RL-RANFIST2	Gain-scheduled Reinforcement Learning Recurrent ANFIS type 2
GSPI	Gain-Schedule PI
GW	Giga Watts
GWEC	Global Wind Energy Council
GWh	Giga Watt-hours
HAWTs	Horizontal Axis Wind Turbines
HJB	Hamilton-Jacobian-Bellman
IDHP	Incremental Model-based Dual Heuristic Programming
IPC	Individual Pitch Control
IRENA	International Renewable Energy Agency
LM	Levenberg-Marquardt
LMB	Levenberg-Marquardt Back-propagation
LQR	Linear Quadratic Regulator
LSTM	Long-Short-Term Memory
MDP	Markov Decision Process
MPPT	Maximum Power Point Tracking

MSE	Mean Square Error
MIMO	Multi-Input-Multi-Output
MPC	Model Predictive Control
MPPT	Maximum Power Point Tracking
MW	Mega Watt
NREL	National Renewable Energy Laboratory
O-P	Only Positive
OC3-Hywind	Offshore Code Comparison Collaboration involves the modeling of an offshore floating wind turbine
OpenFAST	Open-source Fatigue-Aerodynamic-Structure-Turbulence software
PM	Phase Margin
P-N	Positive-Negative
PMSG	Permanent Magnet Synchronous Generator
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
PSO-P	Particle Swarm Optimization Policy
Q-Learning	Quality Learning
Q-Networks	Quality Networks
RBF	Radial Basis Function
RBF-NN	Radial Basis Function Neural Network
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
SGD	Stochastic Gradient Descent
SSE	Sum of Square Error
STD	Standard Deviation
RNN	Recurrent Neural Network
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
VSWT	Variable-Speed Wind Turbines
WECS	Wind Energy Conversion Systems
WTs	Wind Turbines

List of Symbols

$\boldsymbol{A_{avg_n}}$	Average state matrix of linearized WT model at n^{th} operating point
$\boldsymbol{B_{avg_n}}$	Average input matrix of linearized WT model (Continuous system)
$\boldsymbol{C_{avg_n}}$	Average output matrix of linearized WT model (Continuous system)
$\boldsymbol{x_n}$	WT system states vector (Continuous system)
u_{cpc}	Collective pitch control input
y	Perturbations in generator speed output
T_n, T_m	WT linearized model transfer functions at average wind speeds (n, m)
$\overline{T_n}$, $\overline{T_m}$	The conjugates of T_n and T_m.
$D(T_n, T_m)$	Maximum gap distance between two transfer functions
$\|\|.\|\|_\infty$	Infinite Norm (largest component in the vector representing the difference between (T_n, T_m) in frequency domain)
S	system's state space
A	The action space
(s_k, a_k, s_{k+1})	(current state, current action, and next state) at time k
T	The system's transition function
$R(s_k, a_k, s_{k+1})$	Reward function in stochastic environment (MDP)
$\bar{R}(s, a)$	Reward function at case of deterministic policy
$T(s_k, a_k, \cdot)$	Probability density transition function, dot represents the following unknown probabilistic states
$\bar{T}(\mathrm{s}, a)$	Deterministic system dynamics
$\Pr(s_{k+1} \| s_k,\ a_k)$	the probability of being in s_{k+1} based on s_k and a_k.
μ	The agent's policy that takes actions
$\mu_d(s)$	Deterministic policy (function maps the environment states (s) to a specific action)
$\mu_s(a\|s)$	Stochastic policy (the probability of taking action a in state s)
$\mu_s(s, \cdot)$	Probability distribution over action-space A given a state s
μ^*	Optimal Policy
$\hat{\mu}^*$	Approximated optimal policy (in Q-iteration algorithm)
γ	Discount factor
a	Current action
$\boldsymbol{a'}$	Next action
s	Current state of a system
s'	Next state of a system
γ	Discount factor to future rewards
$V(s)$	State-value function (estimated return of being in state s)
$V^{\mu}(s)$	the estimated return considering the random transitions acquired from adhering to the policy μ

$V^*(s)$	optimal (maximum) V-function
$\mathbb{E}_{s' \sim T(s,\mu(s),\cdot)}$	Expectation of a discounted future reward according to the next estimated state being in the probability distribution of the transition function
$\boldsymbol{Q^{\mu}(s,a)}$	Quality - function represent the expected return of taking a specific action from a given state and following a policy
$Q^*(s,a)$	Optimal Quality-function
$\hat{Q}^*$	Approximated Optimal Q-function (in Q-iteration algorithm)
$\mathbb{Q}$	State-weights matrix
$\boldsymbol{\mathcal{R}}$	Action-weights matrix
$\mathbb{R}$	Real numbers
ϵ	Probability of choosing random actions
$\underset{a}{\textbf{argmax}}$	Choice of action (a) that achieves the maximum expected return
ϕ^{μ}, $\phi^{\mu'}$	Behavior actor (Policy), and target actor neural networks parametrization (weights)
ϕ^{Q}, $\phi^{Q'}$	Behavior Critic (Q-network), and target critic neural networks parametrization (weights)
ϕ^{V}	State-value-function parametrization
$L_t(\phi^Q)$	Loss function of Q-network that vary at each iteration t which represents the difference between estimated and target Q-values
$Q'(s,a\|\phi^{Q'})$	Target Q-network values as function of state-action pairs and parameterized by $\phi^{Q'}$
$\mu_d(s\|\phi^{\mu}) = \mu(s\|\phi^{\mu})$	Deterministic actor-network as function of state and parameterized by ϕ^{μ}
$\mu'(s\|\phi^{\mu'})$	Target deterministic actor-network as function of state and parameterized by $\phi^{\mu'}$
(s_x, a_x, r_x, s_{x+1})	Random mini-batch tuples of transitions from the experience replay buffer
y_x	Target expected future return at case of random mini-batch from replay buffer
a_{x+1}	Action that maximizes the expected reward from the target network
$U(D)$	Distribution of the experience replay buffer
$\mathbb{E}_{(s_x,a_x,r_x,s_{x+1})\sim U(D)}$	The expected value of selecting random samples (tuples) from the distribution of the experience replay buffer
$\nabla_{\phi_t^Q}$	Gradient descent with respect to parameters ϕ^Q at time (t)
α	Learning rate
K_p, K_i	PI controller parameters
u_{ss}	Steady state look-up table control action
u^*_{FMPC}	Optimal collective pitch control action extracted from the FMPC
$\boldsymbol{u(k)}$	Control action u at sampling instant (k)
$\boldsymbol{x_{dn}}$	WT system states vector (Discrete linearized model)
$u_{u_{FMPC}}$	Perturbation in the collective pitch control
$\boldsymbol{A_{dn}}$, $\boldsymbol{B_{dn}}$, $\boldsymbol{C_{dn}}$	Matrices of the discretized linear models.

$\boldsymbol{A_e}$, $\boldsymbol{B_e}$, $\boldsymbol{C_e}$	Equivalent WT linearized model matrices output from the fuzzy system
$\boldsymbol{M_n}$	Represents any of the matrices $\boldsymbol{A_{dn}}$, $\boldsymbol{B_{dn}}$ or $\boldsymbol{C_{dn}}$ at $n = 2, 4, 6$
$\boldsymbol{M_e}$	The output matrix from the fuzzy system, which represents $\boldsymbol{A_e}$, $\boldsymbol{B_e}$ $\boldsymbol{C_e}$.
N_p , N_c	Prediction and control horizons
$\boldsymbol{y_F}$	predicted output vector
$\boldsymbol{F}$	state prediction matrix (state-to-output mapping matrix)
$\boldsymbol{\Phi}$	Input prediction matrix (input-to-output mapping matrix)
$\boldsymbol{\Phi_1}$	Input prediction matrix relating to the inputs from $u_{FMPC}(k)$ up to $u_{FMPC}(k + N_c - 1)$
$\boldsymbol{\Phi_2}$	Input matrix relating to the inputs from $u_{FMPC}(k + N_c)$ up to $u_{FMPC}(k + N_p - 1)$
$\boldsymbol{u_F}$	Sequence of predicted control actions vector over N_c
$\boldsymbol{u_F^*}$	Optimal sequence of predicted control actions vector
J	Cost function (in context of FMPC design part)
$\boldsymbol{\bar{R}}$	diagonal matrix with shape = $N_p * N_p$
z	Tunable parameter reflecting the penalization on the control action
u^{max} , u^{min}	maximum and minimum permissible pitch angles
α^{max} , α^{min}	maximum and minimum permissible pitch angle rates
$\boldsymbol{T_s}$	Sampling time
$\boldsymbol{L}$, $\boldsymbol{M}$	Constraints matrices
$\boldsymbol{\lambda}$	Lagrange multipliers vector
$\boldsymbol{\lambda}^*$	optimal solution of the Lagrange multiplier vector
$\boldsymbol{x}^{(i)}$	features (states) input vector to CNN at sample (i)
N_S, N_X	Number of samples and number of features
N_{l-m} , N_l	Number of neurons in layers $[l - m]$ and $[l]$
$\boldsymbol{X}$	Features (inputs) (state) layer matrix
$\boldsymbol{Z}^{[l]}$	Linear output matrix of the hidden layer $[l]$
$\boldsymbol{W}_{\boldsymbol{X}}^{[l]}$	weight matrix connecting input layer matrix $\boldsymbol{X}$ to the neurons of layer $[l]$
$\boldsymbol{b}^{[l]}$	Bias vector at layer $[l]$.
$\boldsymbol{A}^{[l]}$	Output matrix of the tanh activation function at layer $[l]$
$\boldsymbol{W}_{\boldsymbol{A_{l-m}}}^{[l]}$	weight matrix connecting neurons of layer $[l]$ with the neurons of layer $[l - m]$
$\boldsymbol{u_{cpc}}$	Actual (Target) collective pitch output vector from FMPC
$\boldsymbol{\hat{u}_{cpc}}$	Estimated collective pitch output vector from CNN at all samples
$\boldsymbol{w}$	flattened vector that contains all CNN parameters (weights and biases)
$\boldsymbol{\Delta w}$	CNN weights updates
$\boldsymbol{w}^*$	optimal CNN weights
$e_i(\boldsymbol{w})$	residual error between actual and estimated CPC at sample i based on $(\boldsymbol{w})$

$L_{CNN}(\boldsymbol{w})$	Sum of square errors loss function of the CNN
$\nabla \boldsymbol{L_{CNN}}(\boldsymbol{w})$	Gradient matrix of the loss function
$\nabla^2 \boldsymbol{L_{CNN}}(\boldsymbol{w})$	Hessian matrix of the loss function
B	Mini-batch size
$P \equiv N_X$	Total number of input features to the CNN
M	Total number of parameters in the CNN
$\mathbf{J}(\boldsymbol{w})$	Jacobian matrix of residual errors
ζ	Damping factor/adaptive learning rate in LMB algorithm
ρ	Factor greater than one to control the momentum of learning in LMB
R_B	Experience replay buffer memory storage
J_t	Objective function representing the cumulative expected return (in the context of F-DDPG design part)
θ_{DT}	Torsional displacement of the drive-train of WT system
$\Delta\omega_{rotor}$	Error in rotor speed of WT system
ω_{DT}	Torsional speed of the drive-train of WT system
P_{Gen}	Generator output power of WT system
P_{rated}	Generator output rated power = 5000 kW
ω_{rated}	Roto speed rated value = 1.23 rad/s (11.75 rpm)
$\left\|\|\cdot\right\|\|$	Normalized absolute error
f_m, s_m	First and second moments values of ADAM optimization
$\hat{f}_m, \hat{s}_m$	Corrected first and second moments values of ADAM optimization
δ	Factor to avoid division by zero
E	Number of episodes
T	Time steps per episode
T_s	Sampling interval
t	Time step
a_{min}	Minimum permissible pitch angle
a_{max}	Maximum permissible pitch angle
α_μ	Learning rate of actor network
α_Q	Learning rate of critic network
λ_μ	L2 Regularization factor of actor network
λ_Q	L2 Regularization factor of critic network
β_1	Exponential decay rate for 1st moment estimates
β_2	Exponential decay rate for 2nd moment estimates
τ	Target network smooth factor
R_B	Experience buffer samples size
$O_{(t_0=1)}$	Initial noise value = Initial action value
ξ	Noise process mean

θ	Noise model mean attraction constant
$\sigma_{(t_0)}$	Initial noise standard deviation
HLT	Half-life-time Samples
σ_δ	Noise standard deviation decay rate
σ_{min}	Minimum standard deviation
K	K-step lookahead
$R(t)$	Random number (with mean = 0, standard deviation =1)
$W_{(t)}$	Wiener process
C_i	The control action exerted from i^{th} DDPG trained-agent
C_f	Fuzzy collective pitch control law of F-DDPG controller

Table of Contents

List of Tables

List of Figures

Abstract

Wind turbines (WTs) exhibit complex, nonlinear behaviors and uncertainties, influenced by aerodynamic, mechanical variables, and variable wind conditions, with added challenges from turbulence and wind shear in the constant power region (region 3). The core of operating WTs in region 3 condition is to maintain the generator power and speed at the rated values. Addressing these complexities requires comprehensive insights into the system's nonlinear dynamics and uncertainties. This thesis introduces innovative, model-free AI-based strategies for collective pitch control (CPC) that successfully regulate generator speed and output power in region 3, aiming to keep them at their rated levels with minimal fluctuations during high wind conditions.

The proposed controllers regulate generator speed, improve power output, and reduce fluctuations while accommodating system uncertainties, nonlinearities, unmodelled dynamics, and pitch limits. The disparity between WT dynamics due to wind speed perturbations and uncertainties is measured using a gap-metric criterion to ensure robustness. The first proposed controller design adopts a supervised learning concept. It relies on the neural network as a function approximator between the state input and CPC output data. The network's topology is called cascaded-forward neural network (CNN). This topology is chosen for its effectiveness in capturing the linear and nonlinear relations between the input and output data, unlike the conventional fully-connected neural networks which capture only the nonlinear relations. The objectives are to design an optimal and computational efficient AI-controller. The optimality is achieved through training from the data provided by a fuzzy-based model predictive controller (FMPC). The FMPC is chosen to provide highly efficient samples. Also, the optimality is achieved by managing the WT system with the FMPC controller across various wind speed conditions, encompassing the entire scope of operation in region three. On the other hand, to achieve a computationally efficient controller, the number of layers and neurons of the neural network chosen for training is minimum. Regarding the training phase, the cascaded-forward neural network-based controller (CNN-C) undergoes offline training, processing input states using forward propagation to predict the CPC values. The loss function is calculated as the sum of squared errors between these predictions and actual CPC values from sampled data. This function is then optimized using the Levenberg-Marquardt back-propagation (LMB) method to reduce errors. Once optimized, the CNN-C is deployed in real-time simulations of the WT system.

The second proposed controller design adopts the deep reinforcement learning (DRL) approach, which mixes reinforcement learning and deep learning. It relies on online learning through the interaction of the agent (controller) with the WT model. The chosen algorithm is called the deep deterministic policy gradient (DDPG) algorithm. The algorithm is explicitly chosen due to its ability to handle high-dimensional, system uncertainties, and unmodelled dynamics of the WT model. The training phase includes training of six agents in a medium-fidelity WT environment at different mean wind speeds to ensure the controller's robustness. Initially, imitation learning is used for efficient sample collection to fasten training convergence. Afterwards, each agent learns through interaction with the environment to gain optimal experience. At the deployment phase, the pitch control outputs from the multi-trained agents are processed by a fuzzy system to have smooth transitions under different operating conditions. The resulting controller, called fuzzy DDPG (F-DDPG), is deployed in real-time simulations of the WT system to obtain the optimal CPC.

Simulation tests are conducted using the high-fidelity WT model based on Open-FAST to assess the effectiveness of the proposed control approaches against conventional control methods. Open-FAST is a comprehensive aero-hydro-servo-elastic simulation tool developed by the National Renewable Energy Laboratory (NREL) and MATLAB/Simulink. This investigation evaluated the capabilities of a high-fidelity 5-MW horizontal-axis onshore WT model and an "OC3-Hywind spar buoy" floating offshore WT model. The performance of the CNN-C was compared with that of the FMPC and gain-schedule PI (GSPI) controllers.

Furthermore, the efficacy of the F-DDPG controller is measured against the GSPI, Linear-Quadratic-Regulator (LQR), and single-DDPG-agent controllers. Then, a comprehensive performance analysis of all controllers is delivered. The results revealed that the newly proposed AI-based controllers outperformed the others in different operational aspects.

Chapter 1 : Introduction

Wind turbines (WTs) have emerged as pivotal electrical generation assets within the global shift towards renewable and sustainable energy paradigms, reflecting a commitment to mitigating climate change and promoting a clean environment. These sophisticated electromechanical systems offer substantial benefits in their minimal environmental impact, operational reliability, and potent energy generation capabilities. According to the International Renewable Energy Agency (IRENA), the cumulative installed capacity of wind turbines globally reached approximately 898.856 gigawatts (GW) by the conclusion of 2022 [1]. Furthermore, Global Wind Energy Council (GWEC) projections for 2023-2027 anticipate an aggressive expansion, with an estimated 550 GW of new installations, translating to an average annual increment of 110 GW [2]. This forecast underscores the strategic importance of wind energy in addressing future energy demands sustainably. The economic viability of wind turbines is a critical consideration, predicated on maximizing annual energy output, measured in gigawatt-hours (GWh). Achieving this requires the strategic siting and design of wind turbine installations and the implementation of advanced control systems. These systems are essential for maintaining operational stability and optimizing energy capture efficiency, thereby enhancing the overall performance of wind turbines. Through the integration of cutting-edge technologies and control methodologies, wind turbines are set to play an increasingly significant role in the global energy mix, offering a path towards a more sustainable and cleaner energy future.

The control systems come in various types, including pitch control, yaw control, and torque control. Each is designed to address specific aspects of turbine operation. Pitch control adjusts the angle of the blades to control the rotor speed and optimize power output, yaw control aligns the turbine with the wind to maximize energy capture, and torque control manages the generator speed to enhance efficiency and protect the turbine from excessive stress. The pitch angle control is the cornerstone of the thesis.

The chapter organization is as follows. The problem definition and the motivation for the thesis are outlined in Subsection 1.1. The aims and objectives of designing sophisticated AI-based controllers in the thesis are outlined in Subsection 1.2. Following that, the literature review is introduced in Subsection 1.3. The contributions in this thesis are presented in Subsection 1.4. The thesis organization is finally detailed in Subsection 1.5.

1.1. Problem Definition and Motivation

The WECS energy harvesting depends on the available hub-height wind speeds. According to these speeds, the WECS operation is divided into three main regions of operation. These regions are detailed and depicted in Chapter 2, specifically in subsection 2.1.4. In brief, region 1 specifies the range of low wind speeds up to cut-in value, where the WT starts to operate by speeding up the rotor with no power generation. In region 2, the wind speeds are entrapped between cut-in speed and rated

wind speed where the rotor reaches its rated speed. The objective of control design in this region is to maximize the power extraction and achieve the optimum power coefficient from the WT. In region 3, called the constant power region, the wind speeds lie between rated wind speed and cut-out speed. The primary motivation behind the control design at this region is to limit the power captured by the WT to its rated power while maintaining the operation at the rated rotor speed [3]. This goal can be achieved by an optimal pitch control mechanism that effectively regulates the rotor loads and aerodynamic power without exceeding the safety constraints and limits of the design [4], [5]. The blade inertia determines the velocity of blade rotation and the robustness of the pitch mechanism. Consequently, the maximum pitch rate of the WT model used is ($8°/sec$) [3], [6].

In region 3, pitch control consists of three types: individual pitch control (IPC) [7], cyclic pitch control, and collective pitch control (CPC). IPC adjusts the pitch angle of each blade individually to reduce asymmetric loads resulting from the turbulence on the WT rotor [7]. Cyclic pitch rotates each blade to be like others at the same azimuth angle of the rotor, and it could reduce the dynamic loads on the WT. CPC is responsible for adjusting all blades simultaneously to the same pitch angle while achieving the rated power extraction and rated rotor speed from the WT.

This study focuses on the CPC system in region 3 operation mode, satisfying the pitch angle and pitch rate constraints. However, pitch control of WTs in region 3 encounters some difficulties that must be addressed. These difficulties could be summarized as follows:

1. Wind turbines in region three are subjected to rapid and unpredictable changes in wind conditions. High-speed winds, gusts, and turbulence can cause significant and sudden changes in aerodynamic forces, affecting the controller's stability.
2. The aerodynamic complexity of wind turbines is highly non-linear. There are uncertainties in turbine model parameters. The variations in wind speeds, changes in blade pitching, and structural flexibilities add further nonlinearities and uncertainties, making it challenging to design an optimal controller.
3. The regulation of WT operation in region 3 is challenging. This is summarized as operating at the rated generator speed and maximizing output power with minimum fluctuations.
4. The pitch control action has limitations in terms of speed and precision.

These complexities require highly advanced control strategies to ensure system efficiency and stability. Designing an efficient pitch controller demands an accurate WT system model that comprehensively encapsulates the highly nonlinear dynamics, potential instabilities, and unpredictable nature of wind patterns in this region. Achieving such a detailed and precise model is a daunting task, crucial for overcoming the inherent difficulties in controlling WTs under these demanding operational conditions. The motivation behind our study is to design an effective model-free AI-based optimal pitch control system that considers all mentioned complexities and difficulties to ensure controller robustness, stability of operation, low computational cost, and optimized energy harvesting.

1.2. Objectives

This thesis aims to design and investigate AI-based pitch angle controllers to operate in region 3 of a 5-MW high-fidelity WT optimally. The design goals include fastening the learning time, robustness against stochastic winds, regulating power output, maintaining rated generator speed, reducing fluctuations, handling high nonlinearities and model uncertainties, and enhancing optimality. In order to achieve these goals, two AI controllers will be proposed. The first controller is based on the supervised learning concept. Its primary goal is to generalize the fuzzy model predictive controller's (FMPC) performance and minimize the computational cost of execution. The second controller is a modified version of a deep deterministic policy gradient (DDPG), a deep reinforcement learning-based collective pitch controller, where the training is initially guided by an FMPC [8].

1.3. Literature Review

This section provides an overview of the existing literature on AI-based control systems for wind turbines. It includes a concise review of the objectives for controllers operating in region 2 and an in-depth survey of pitch control strategies in the constant power region (region 3).

In region 2 operation mode, the wind speeds lie between the cut-in and rated values. The objective of the control system in this region is to maximize power extraction. This objective is attainable by applying maximum power point tracking (MPPT) algorithms. The maximum power extraction depends on the wind speed, with the generator speed needing to hit a precise value for optimal power output. The MPPT algorithm tracks this value to maximize the generator output power. The conventional and advanced MPPT principles are detailed in [9] and [10]. In [11] and [12], the authors develop a supervised learning fully-connected layer neural network controller for MPPT. In [13], the authors develop an MPPT using unsupervised neural networks and genetic algorithms. In [14], the authors develop an MPPT using a PI controller along with neural networks for system uncertainties compensation. In [15], the authors develop a real-time adaptive AI-based optimal perturbation and observe for MPPT. In [16] and [17], the authors leverage the power of reinforcement learning in MPPT applications of WECS. For a summarized literature review on the latest MPPT AI-based controllers, the authors in [18] detail a literature review on the recent AI-based MPPT controllers in WT applications.

In region 3 operation mode, the wind speeds lie between the rated and cut-out values. The controller's primary objective in this region is to efficiently stabilize the generator speed and power at their rated values. This objective is attained by real-time adjustment of the pitch angles of the WT blades.

Nowadays, the focus is on developing cutting-edge artificial intelligence and reinforcement learning (RL) control algorithms for WT operation. The survey related to the proposed fuzzy deep deterministic policy gradient (F-DDPG) controller relies on [20] – [34]. In addition, the literature survey related to the proposed cascaded-forward neural network controller (CNN-C) relies on [35] – [51].

In [19], the authors develop an adaptive model-based controller that uses radial-basis function neural networks and Lyapunov stability for maximum power point tracking and pitch angle control with no tracking error. The adaptive neural network mitigates the high disturbance and uncertainties in the dynamics of the WT. The controller performance is tested on low-turbulence wind profiles for 400 seconds duration.

In [20], the authors focus on pitch angle control using DDPG in tuning the parameters of a nonlinear backstepping controller. The Lyapunov analysis has been conducted to ensure the controller's stability. The controller performance is validated in software-in-loop and hardware-in-loop tests for 35 seconds. The software WT model used is of low fidelity as the authors neglect the towers, platform, and sophisticated aerodynamics.

In [21], the authors use adaptive dynamic programming with a temporal difference technique to develop a CPC. The controller relies on the actor and critic networks concept with real-time adaptation of their parameters. The actor network is responsible for taking the pitch control action. The critic network is responsible for evaluating the actor network's performance. The optimization of the networks' parameters is chosen as the backpropagation with the gradient descent concept. A simplified WT model for developing and testing the controller performance is used with a 100-second testing duration. Furthermore, a moderate turbulence wind speed profile is used for a testing scenario.

In [22], the authors develop a hybrid RL and neural networks pitch controller to maximize power output. The authors discretize the action space to suboptimal actions. The quality of the discretized actions is evaluated using the neural network by mapping the states to the expected rewards of taking each action. The Quality-Learning (Q-learning) algorithm minimizes the error between the expected rewards from the neural network and the actual rewards. The action that achieves the maximum expected reward is chosen as input to the WT. The development and testing of the controller rely on a simplified WT model with a 100-second duration of testing.

In [23], the authors propose a data-based RL pitch angle controller by constructing an augmented time-delay system based on the control input, time delay, and state as an augmented vector and augmented Bellman equation. Through the collected data from the operation of the WT, the authors build a data-driven WT model. The policy-iteration RL algorithm is used to solve the optimal control problem. A simplified WT model with low turbulence is used for testing for 100 seconds.

In [24], the authors develop a model-based RL method that relies on a deep neural network for system dynamics approximation and model predictive control (MPC) for pitch angle control. The authors use neural networks to identify systems. The quadratic formula of generator speed and pitch control action is used as an objective function for optimization purposes.

In [25], the authors develop a hybrid RL-based control algorithm with a conventional PID controller for pitch angle control. The initial learning episodes rely on the traditional PID controller, which acts as a learning accelerator for the RL controller, which has no previous experience with the WT system. A 7-kW simple WT model with

a constant low turbulence wind speed profile is used to test the controller for a 100-second duration.

In [26], the authors develop a trust region policy optimization (TRPO) RL pitch angle controller. The TRPO's objective is to account for the system's constraints. The optimal control problem is a multi-input-multi-output (MIMO). The controller receives the system's states and exerts the pitch and torque control actions for the WT model. The controller is tested on a high-fidelity NREL 5-MW WT model for 120-second duration. The results reveal the low-efficiency operation of the designed controller as the output power is almost below the 4-MW level.

In [27], the authors develop a hybrid controller using deep learning for wind speed forecasting fed to a fuzzy logic controller. A deep recurrent long-short-term memory neural network (LSTM) is used for its capability to predict wind speed based on memorized previously forecasted wind speeds. The controller performance is compared to a fuzzy logic controller with an anemometer to measure the current wind speed. The controller performance is validated on a simplified WT model for 350 seconds. Similar ideas regarding the wind speed estimation using neural networks are introduced in [28] and [29].

In [30], the authors introduce an RL control approach to merge individual pitch control (IPC) with collective pitch control (CPC), targeting load mitigation and efficient power management. An incremental model-based dual heuristic programming (IDHP) is crafted for IPC to diminish structural loads by incorporating the dynamics of floating offshore wind turbines learned in real-time, thereby transitioning the control strategy to a data-driven framework that minimizes reliance on theoretical models. This approach simplifies the design by requiring the learning of only essential parts of system dynamics, which enhances the design's simplicity and accelerates the learning process.

In [31], the study looks at how to manage variable-speed WTs, focusing on two main parts: the blade pitch and the generator torque, which work together to convert wind into electricity. Unlike earlier methods that treated these parts separately, this research uses Actor-Critic RL to adjust both parts simultaneously, considering how they affect each other for the best overall performance. The new control setup can fine-tune settings for different situations, and its effectiveness was checked using the NREL OpenFAST 5-MW WT simulator.

In [32], [33], the authors propose a pitch angle control strategy for wind turbines designed to accommodate the stochastic nature of external wind speeds, time-variance in unit parameters, and the system's nonlinearity using a reinforcement learning algorithm. It employs an Actor-Critic framework, utilizing an radial basis function (RBF) neural network to handle continuous input and output spaces efficiently. This algorithm dynamically optimizes control parameters to adapt to changing environmental conditions.

In [33], the study introduces an RL-inspired pitch control system for WTs, incorporating components such as a state estimator, a reward mechanism, a policy table, and a policy update method. New reward strategies aimed at reducing generator output power deviation from rated value are developed to enhance WT efficiency,

introducing two types of rewards: "only positive" (O-P) and "positive-negative" (P-N), which are analyzed in the context of the exploration-exploitation balance, epsilon-greedy methods, and learning convergence specific to WT control. The study thoroughly examines how these reward strategies impact the controller's performance and learning speed. In addition, it compares the RL-based controller's effectiveness with that of a traditional PID regulator on a small wind turbine.

In [34], the authors present a control approach that utilizes passive reinforcement learning RL optimized by a particle swarm optimization policy (PSO-P) to manage a type-2 adaptive neuro-fuzzy inference system (ANFIS) with unsupervised clustering for the pitch angle control of an actual WT. The control strategy employs a gain-scheduled RL recurrent ANFIS type 2 (GS-RL-RANFIST2) pitch angle controller to keep the rotor speed constant at its rated value and smooth out power output while enhancing the pitch angle system's performance.

In [35], the authors introduce a brain emotional learning-based intelligent controller (BELBIC) designed for precise pitch control of a 5-MW WT, drawing inspiration from the mammalian limbic system for model-free learning that adapts to system changes and uncertainties. The BELBIC's self-learning feature effectively manages the wind turbine's nonlinearities and disturbances, ensuring accurate pitch angle maintenance even in unexpected wind scenarios. Performance and resilience under various wind conditions, including gusts and random winds, were tested in MATLAB/Simulink, with outcomes demonstrating BELBIC's superiority over traditional fuzzy-PID and gain-scheduling PI control methods.

In [36] and [37], the authors develop a fully connected neural network pitch angle controller with the radial basis function being the activation function. The particle swarm optimization (PSO) evolutionary technique is used in [37] to choose optimal data being fed to the network. The neural networks are used to determine the optimal gain parameters of a PI controller in [37] and [38] While in [36] and [39] the neural network is used directly to exert the pitch control output.

In [40], the study develops an intelligent predictive model for WT parameter variables and control using a deep extreme learning machine (DELM), simplifying higher-order nonlinear models through unsupervised hierarchical feature extraction and layer-by-layer encoding. Utilizing an extreme learning machine (ELM) for accurate feature-to-output mapping minimizes information loss, with the output guiding pitch control via an radial basis function (RBF) neural network. Simulation results demonstrate the algorithm's effectiveness in load mitigation, offering a valuable reference for wind turbine controller design.

In [41], the paper introduces an adaptive neural pitch angle control strategy for variable-speed wind turbines (VSWT) within the pitch control region, aiming to stabilize rotor speed and generator power despite external disturbances without requiring detailed system parameters and aerodynamic knowledge. The strategy includes increasing the system dynamics order through a filtered regulation error, transforming the non-affine VSWT model into an affine control problem for a more straightforward application of feedback linearization, ensuring continuous control signals, and reducing mechanical stress on pitch systems. It leverages an online

learning approximator for estimating unknown aerodynamics and employs a high-gain observer for rotor acceleration estimation, eliminating the need for extra sensors.

In [42], the study proposes a hybrid pitch-control strategy that merges a lookup table with a radial basis function (RBF) neural network, where the neural network corrects the lookup table's mapping errors, and the table aids the network's learning process, resulting in improved efficiency over individual techniques. Comparative analyses against standalone neural control, PID regulation, and their combination with a lookup table show that the hybrid approach significantly reduces output power errors.

In [43], the authors introduce a pitch angle controller based on a Feed Forward Back Propagation Neural Network (FFBP-NN) to reduce power fluctuations in grid-connected wind generation systems. It highlights that the controller optimally tracks the WT power at below-rated wind speeds to smooth out power output, while at above-rated speeds, it employs conventional power regulation methods for stabilization. The FFBP-NN controller utilizes online training with the Levenberg–Marquardt (LM) algorithm, updating the neuron connection weights through back propagation to enhance performance. A similar idea exists in [44] regarding the grid-connected power regulation of wind turbine using neural networks.

In [45], the study develops an adaptive sliding mode controller to maintain the power output of a wind turbine at its constant rated level by regulating the turbine's speed. It incorporates a Radial Basis Function Neural Network (RBF-NN) to accurately estimate the nonlinearities and uncertainties in the aerodynamic model, including the power coefficient approximation. The research introduces a continuous function as an alternative to the traditional sign function to mitigate the chattering effect commonly associated with sliding mode control. This function ensures that the closed-loop system's convergence is mathematically validated.

In [46], the authors introduce a neural network-based control strategy to adjust the wind turbine system's blade pitch angle and rotational speed. This acknowledges the system's affine model and the aerodynamic model's indication of a strong coupling between turbine speed and pitch angle. Given the limitations on pitch angle and rate, alongside the presence of an unknown actuator dead-zone, the wind turbine is treated as a nonlinear system. To address these complexities, a neural network controller is employed for system estimation, complemented by a static network serving as a compensator for the actuator's unknown dead-zone behavior. A similar idea exists in [47]. This method includes a system identification process within the controller that adjusts the pitch angle at high wind speeds to maintain the turbine's rated output power. Utilizing the RBF neural network for system identification, the approach dynamically adjusts PID parameters in real-time based on the system's identified characteristics and a specified learning rate.

In [48], the article introduces a hybrid control system combining a fuzzy sliding mode controller for optimizing the speed of a permanent magnet synchronous generator (PMSG). It also presents an advanced recurrent neural network (RNN) for controlling the turbine pitch angle, featuring online training capabilities. Utilizing the back-propagation algorithm, the RNN controller's parameters are fine-tuned for accurate pitch angle control. The PMSG speed controller employs maximum power point tracking for efficient energy capture from varying wind speeds below the rated

threshold. The sliding mode controller also incorporates a fuzzy inference mechanism to estimate and compensate for system uncertainties accurately, ensuring robust performance across different operational conditions.

In [49], this research introduces an intelligent pitch control system for wind turbines equipped with doubly fed induction generators, leveraging an adaptive neural network fine-tuned through Differential Evolution (DE) optimization. The initial weights of the neural network controller are derived from DE-optimized training data, with subsequent real-time adjustments based on changes in system output.

In [50], the authors develop an RBF network-based neural controller for managing the pitch of wind turbines, utilizing unsupervised learning algorithms for its operation. The RBF network inputs include the discrepancy between actual and desired power output and its rate of change, with the error's integral driving the learning mechanism. Comparative analyses demonstrate the neuro-controller's superior performance over traditional PID regulation for a small wind turbine, highlighting the impact of RBF network settings, wind velocity, learning parameters, and control intervals on the system's behavior.

1.4. Contributions

1.4.1. The proposed cascaded-forward neural network controller (CNN-C)

Neural networks play a crucial role in enhancing the efficiency and reliability of wind turbines by serving as pitch angle controllers. These controllers are vital for adjusting the angle of the WT blades in real-time, optimizing the capture of wind energy, and reducing mechanical stress on the turbine during varying wind conditions. Neural networks, having the ability to learn and adapt from data, offer a significant advantage in managing the complex, nonlinear dynamics of WT operation. They can predict optimal pitch angles based on incoming wind speed data, improve power generation efficiency, and contribute to the longevity of turbine components.

However, applying neural networks as pitch angle controllers in WTs has challenges. A primary challenge is the need for substantial training data to model the highly variable wind environment and turbine responses accurately. This requirement can make the initial setup and ongoing adaptation of the neural network resource-intensive. Furthermore, neural networks have a "black box" nature. This nature could complicate the understanding and diagnosis of control decisions, potentially making troubleshooting and fine-tuning the control system more complex. Additionally, real-time processing demands can be high, requiring significant computational resources to ensure timely and accurate pitch angle adjustments.

Despite these limitations, using neural networks to control wind turbines' pitch angle represents a forward-looking approach to maximizing renewable energy production. With continued advancements in computational power and machine learning techniques, it is expected that the efficacy and efficiency of neural network-

based controllers will further improve, making them even more integral to the operation of modern wind turbines.

The proposed CNN-C is innovatively introducing a new design scheme for the neural network controllers, unlike the previous work, which relied mainly on a hybrid control approach where both neural networks and traditional control approaches are used simultaneously. In addition, the computational cost of training and operation is neglected in these previous approaches.

This study presents a neural network-based controller that relies on supervised learning and imitation learning concepts. The CNN is chosen for the design due to its ability to capture the linear and nonlinear relations between the input and output data [51] . The approach collects system-state input and CPC output data from operating a Fuzzy-MPC on a medium-fidelity WT model at different operating conditions. Consequently, these data are fed to train the CNN-C offline. The loss function, which must be minimized, consists of the sum of square error between the actual and expected control output.

The CNN-C contributions are summarized through the following five aspects:

1. The CNN-C evolves from the supervised learning and imitation learning concepts. The FMPC efficiently provides a significantly low number of samples used for CNN-C offline training.
2. The CNN topology is straight-forward and simple to design.
3. The training data encompass the primary states that affect the controller's performance and efficient samples that cover all wind speeds in region three. Thus, there is no need to include the fuzzy logic system that manipulates the system parameters being fed to the controller, as in the case of the FMPC.
4. The algorithm is computationally efficient. It exhibits the shortest training time and fastest execution time.
5. The proposed method represents significant enhancement and generalization in the performance compared to the FMPC. The controllers are tested on a high-fidelity offshore WT model where hydrodynamics and wave perturbations are considered.

1.4.2. The proposed fuzzy deep deterministic policy gradient (F-DDPG) controller

Many reinforcement learning algorithms encounter challenges associated with their use. The first challenge is that applying RL algorithms dependent on discrete action spaces in control systems, especially those that involve complex continuous dynamics like WT, can lead to several issues. The first issue involves quantization errors, which occur when these RL algorithms approximate continuous actions with discrete values. This approximation can introduce inaccuracies, as the actual value of a continuous action may fall between two discrete choices. The second issue is the suboptimal solutions, meaning that the best action might not exist in a discrete set. The third issue is the difficulty in handling high dimensionality, which means that the number of possible action combinations increases exponentially, making the problem computationally infeasible. The second challenge that faces RL is the exploration versus exploitation dilemma. The agent must balance exploiting known rewards and exploring the environment for potentially higher rewards. The third challenge is the low

sample efficiency of RL algorithms, which typically need extensive interaction with the environment to determine an optimal policy. The fourth challenge is creating a suitable reward function that successfully directs the agent to learn the required behavior, which is complex. Unintended agent behavior may result from poorly constructed reward functions. Lastly, RL agents need to generalize from their experience to unseen states. Overfitting to the training states can limit the agent's performance in new, unseen states [52], [53].

The DDPG algorithm is recommended to achieve performance generalization and stabilization in a continuous action space without the need for mathematical modelling equations [54]. It is a simple model-free off-policy RL controller based on deep Q-Networks and actor-critic structure [55]. The term "deep" in DDPG refers to using deep neural networks as function approximators for actor and critic networks. The term "deterministic policy gradient" refers to deterministic action taken by an actor-network that is updated continuously based on policies that conduct gradient ascent on a scalar objective function called the quality (value) function (Q-function) [52] Consecutively, this paper aims to design a modified version of a DDPG deep-RL-based collective pitch controller where the training is initially guided by a fuzzy model predictive controller (FMPC) [8]. The design goals include fastening the learning time, robustness against stochastic winds, maximizing power output, maintaining rated generator speed, reducing fluctuations, handling high nonlinearity and model uncertainty, and improving controller response to actuator limitations.

The DDPG algorithm offers five critical advantages for continuous control: it extends deep Q-learning to continuous action spaces [56], efficiently explores through adding noise to control action, ensures learning stability via actor-critic networks [57], employs off-policy learning that can learn from past experiences stored in a replay buffer to improve stability, and is compatible with complex function approximators like neural networks. However, it also faces challenges such as high sample requirements for convergence because it is a model-free algorithm [52], potential learning instability, policy gradient bias [56], and limited robustness in stochastic environments like wind turbines (WT). Model predictive control (MPC) is a model-based potential candidate for pitch control with four primary advantages [8]. It is robust against disturbances and model errors due to its predictive nature. It can handle various constraints. It utilizes optimization techniques for the best control actions. It offers transparent decision-making [57]. Compared to MPC, DDPG offers unique strengths: it handles the WT's nonlinear dynamics and high-dimensional spaces using deep neural networks, offers online learning and adaptability, and is both cost-efficient and scalable [52]. Unlike MPC, which requires a linear approximation of the model and real-time optimization, a trained DDPG agent is computationally efficient at execution [57].

This study offers a modified version of a DDPG algorithm where six distinct DDPG agents are trained to optimize the agents' collective pitch control action and to construct a robust RL pitch angle controller through the fuzzy logic principle. Each agent is trained under a stochastic wind speed profile with a predetermined average speed to mitigate the issues of learning instability, unmodelled dynamics, and uncertainties caused by high-turbulent wind. The FMPC assists the early learning episodes of training by providing efficient samples to fasten the training convergence. After training (at the deployment phase), the outputs from all trained agents are manipulated by a fuzzy system to construct the optimal CPC action and to ensure

controller robustness at all operating points. The proposed novel control strategy is called fuzzy deep deterministic policy gradient (F-DDPG). This innovative control strategy inherits the MPC's strengths with the advanced capabilities of the modified DDPG to manage WT control effectively in region 3.

The F-DDPG contributions are summarized through the following four aspects:

1. It introduces imitation learning integrated with reinforcement learning [58] in WT control applications to increase the collection of high-efficiency samples to shorten training convergence time [59]. Initially, the FMPC is deployed to enrich the DDPG agent's replay buffer with high-quality samples. Following this, the DDPG agent self-learns using these samples and environmental interactions to improve its performance.
2. The enhancement of the DDPG algorithm's learning process in this study involves two essential modifications. Firstly, it incorporates K-step bootstrapping returns, which allow the algorithm to learn the consequences of its actions over longer horizons more efficiently. This reduces the overestimation of the Q-value challenge that faces the DDPG algorithm [57]. Secondly, changes to the topology of the actor-critic networks have been proposed to reduce the computational costs of learning compared to the standard DDPG algorithm in [52] and [60].
3. A simple fuzzy system is implemented for the CPC actions of the multi-trained DDPG agents to improve the controller's robustness, stability, and performance. This approach has overcome the WTs' high wind uncertainties and stochastic behavior in region three, where wind speeds are intense.
4. Being trained on a medium-fidelity onshore WT model, this study demonstrates the proposed F-DDPG controller's ability to generalize its performance to high-fidelity 5-MW onshore and offshore WT models, closely approximating real-world conditions in region 3. The study emphasizes the proposed controller's compensation of unmodelled dynamics and adaptation to new unconsidered dynamics.

1.5. Thesis Organization

The thesis proposes two novel AI-based pitch angle controllers to optimize the operation of the WTs in a constant power region (region 3). The first proposed controller is based on supervised learning – a Subsection of machine learning - through the provided data from a fuzzy-based model predictive controller (FMPC). Supervised learning is based on a fully connected cascaded-forward neural network. The proposed controller is called CNN-C. The second proposed controller is based on fuzzy logic, deep reinforcement learning, and imitation learning concepts. It is called a fuzzy deep deterministic policy gradient (F-DDPG) controller. The proposed controllers have significantly improved overall performance by maximizing the energy captured, stabilizing generator speed, minimizing fluctuations, enhancing optimality, and effectively reducing computational costs.

Chapter 2 provides essential foundational knowledge for understanding WECS, supervised learning, and reinforcement learning. It delves into the categorization of WECS based on their foundational structure and axis orientation, along with an in-depth look at the critical components of WECS, their operational domains, and comprehensive insights into WECS modeling, analysis, and simulation tools. For supervised learning, the dedicated section outlines crucial guidelines and steps necessary for the effective training of neural networks. Additionally, the dedicated section for reinforcement learning explores RL-based controllers, offering a thorough mathematical analysis of initial reinforcement learning algorithms, their merits and demerits, application challenges, and the progression towards the sophisticated DDPG algorithm, which addresses continuous, infinite state and action spaces.

Chapter 3 outlines the development process of the gain-scheduled PI (GSPI), fuzzy model predictive controller (FMPC), the proposed cascaded-forward neural network-based controller (CNN-C), and fuzzy deep deterministic policy gradient (F-DDPG) controllers. The GSPI section details the adjustment of the PI controller parameters for each operational point. The FMPC section discusses the integration of fuzzy logic, the MPC prediction model, the MPC optimization process, and the control algorithm's framework. The CNN-C section explains how supervised learning is applied in the design of the controller, presenting the CNN structure, its training and optimization methodologies, and the training outcomes. In the F-DDPG section, the approach and architecture of the proposed model-free RL controller are elaborated. These include the design of the actor and critic neural networks, the development of the reward function, a mathematical explanation of the training algorithm, the role of the fuzzy logic system, and the results achieved through training.

Chapter 4 details the simulation outcomes used to assess the effectiveness of the proposed CNN-C and F-DDPG controllers. These controllers are evaluated using detailed simulations on both onshore and offshore WT models to ensure high fidelity. The performance of the CNN-C controller is benchmarked against the FMPC and GSPI controllers, whereas the F-DDPG controller's results are compared with those of the GSPI, linear-quadratic-regulator (LQR), and single-DDPG-agent controllers. The comparison focuses on various operational parameters of the WT in the third operational region, including the rated generator speed and output power, fluctuations in these variables, the pattern of collective pitch control, controller optimality, and their economic impact. Additionally, validation tests for all controllers are carried out to demonstrate the proposed methods' effectiveness and stability across all wind speed operational points. A thorough numerical analysis of the results is also provided, detailing the performance and ranking of each controller based on different operational criteria. This comprehensive evaluation underscores the controllers' operational advantages and their potential impact on WT efficiency and economy.

Chapter 5 discusses the analysis of the results obtained, outlining the primary strengths and weaknesses of the GSPI, LQR, single-DDPG-agent, and FMPC controllers. The discussion then shifts to underscore the significance of the newly proposed CNN-C and F-DDPG controllers in addressing the issues associated with the traditional control methods. Additionally, the critical contributions of each proposed controller are highlighted. The chapter also delves into the limitations of each proposed controller. Moreover, the potential applications and strategies related to the proposed methods are explored, providing a comprehensive overview of their applicability and

impact in the field. Furthermore, this chapter concludes the thesis by summarizing the essential findings and insights gleaned from the study. It outlines recommendations based on the research outcomes and identifies areas for future exploration and development. In addition, the chapter emphasizes the implications of the study's results for the field and suggests directions for subsequent research efforts to build upon the groundwork laid by this thesis.

Chapter 2 : Background

Chapter 2 provides a solid background for the subsequent chapters of the thesis, focusing on two primary subjects: an exploration of wind energy conversion systems (WECS) and their associated control systems. The chapter begins with a comprehensive overview of WECS, detailing the key components, operational regions, and specifics regarding the modeling of wind turbines (WTs) and the simulation of the Open-FAST model employed. Following this, it delves into control systems, highlighting the application of AI-based control strategies for collective pitch control. This section also discusses the rationale for employing the proposed control approaches, including the CNN and the F-DDPG.

This chapter is sectionalized into the following sections. Section 2.1 represents an overview of the WECS and their key components. Section 2.2 presents the system description, parameters, degrees of freedom, and simulation tools. Section 2.3 presents a brief overview of the supervised learning pipeline. Section 2.4 introduces an overview of the RL-based controllers and the evolution of the algorithms up to the building blocks of the DDPG algorithm.

2.1. Overview on WECS

Wind Energy Conversion Systems (WECS) convert the kinetic energy from wind into mechanical power or electricity. These systems use wind turbines to capture wind energy, which is then converted into electrical power through a generator. WECS play a crucial role in renewable energy generation, offering a sustainable alternative to fossil fuels by exploiting the natural wind resources.

2.1.1. Classification of WECS based on foundation

The WECS are of different types, shapes, foundations and orientations. Regarding the foundation aspect, WECS are classified into onshore and offshore models. Onshore wind turbines are located on land and are the most common type of WECS. They are typically more straightforward and cheaper to install and maintain than offshore turbines due to their accessibility. Onshore turbines can often be found in rural or remote areas where they take advantage of the open space and wind flow. However, the wind speed on land can be lower and more turbulent than offshore due to the landscape's natural and man-made features, such as hills, buildings, and trees, which can disrupt wind flow. Onshore wind farms may also face opposition from local communities due to visual impact, noise concerns, and potential impacts on wildlife.

On the other side, offshore wind turbines are installed in large bodies of water, usually on the continental shelf. Offshore turbines are installed based on the depth of the water and energy extraction requirements, as depicted in Figure 2.1. Offshore wind turbines utilize more robust and reliable winds compared to those found on land, resulting in increased energy generation and perhaps higher efficiency [4]. The visual impact is also lessened due to their distance from the coast. However, offshore WECS are more expensive and logistically challenging to install and maintain. They require

specialized ships and technology to handle the marine environment and deep-water installations. The marine environment can lead to quicker wear and tear, increasing maintenance costs and challenges. Offshore turbines also need to withstand strong ocean winds and waves, which may impact marine ecosystems. Despite these challenges, offshore WECS are an increasingly important part of the global renewable energy portfolio due to their enormous power generation potential [61].

In this work, the simulations are conducted on onshore and offshore-floating-spar-buoy models to validate the performance of different control strategies in various operating conditions.

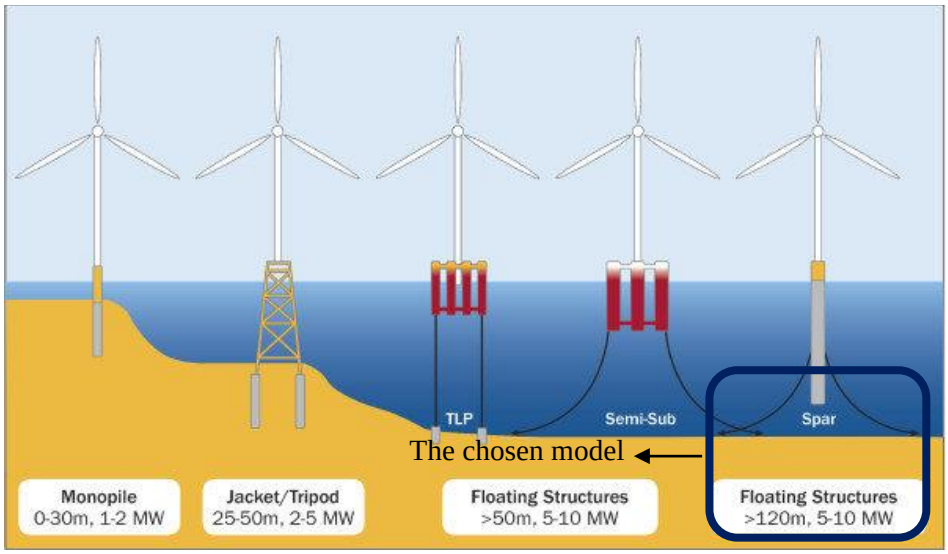


Figure 2.1 Offshore wind turbine models based on their foundation depth. [62]

2.1.2. Classification of WECS based on the orientation of axes

Regarding the orientation of the axis of rotation aspect, the WTs are classified into vertical axis wind turbines (VAWTs) and horizontal axis wind turbines (HAWTs) [63]. The main difference between VAWTs and HAWTs lies in the orientation of their main rotor shaft, as depicted in Figure 2.2. In VAWTs, the rotor shaft is positioned vertically, independent of wind direction. This orientation allows for easier maintenance since the generator and gearbox can be placed on the ground. VAWTs are well-suited for areas where wind direction frequently changes and for sites where tall structures are impractical.

HAWTs have a rotor shaft and generator at the top of a tower, with the rotor blade assembly oriented horizontally to the ground. This design requires the turbine to be oriented into the wind through active mechanisms (like wind sensors and motors) or passively by the wind vane. HAWTs are more prevalent in commercial wind power generation due to their higher efficiency and capacity for larger designs [61].

HAWTs, commonly called propeller-type turbines, dominate the global market, making up 90% of wind turbines in distribution, while the remaining 10% comprises VAWTs, with more than 100 companies providing these models [64]. Thus, our studies in this work are conducted on HAWTs.

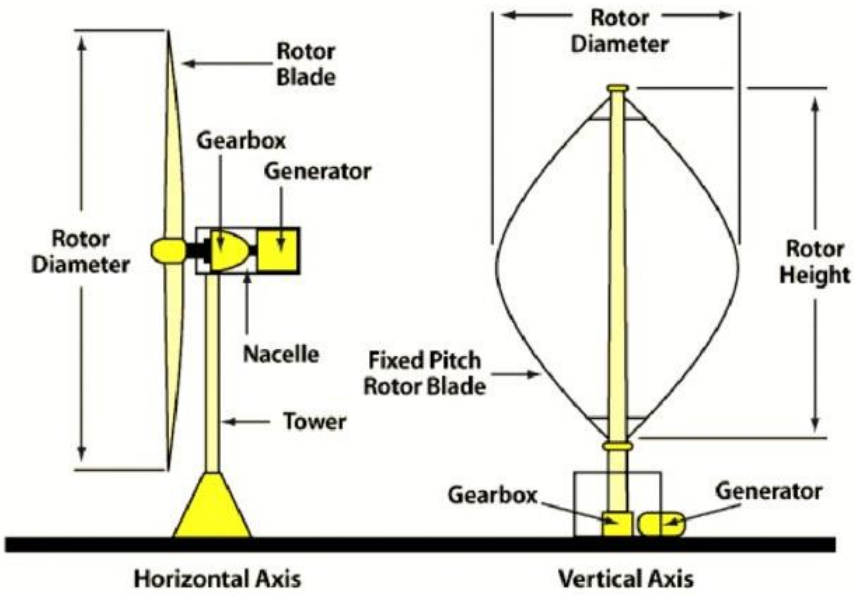


Figure 2.2 The description of the HAWT and VAWT structure. [65]

2.1.3. Key Components of HAWTs

Understanding the architecture of HAWTs is fundamental to appreciating how these turbines capture the wind's kinetic energy and convert it into electricity. From the towering blades to the complex inner workings of the nacelle, each component plays a pivotal role in the turbine's functionality and efficiency.

In this Subsection, every element of the HAWT will be represented to show its function and significance to the system as a whole. This insight is to clarify how these turbines operate. The key components and elements of the HAWTs, as depicted in Figure 2.3, are described as follows:

- **Blades**: These are the large, aerodynamic structures that capture wind energy. The wind causes them to lift and rotate.
- **Rotor**: This refers to the blades and the hub together. It's the part that spins and transfers kinetic energy to the gearbox.
- **Pitch**: The mechanism used to twist the blades along their longest axis controls their angle relative to the wind to maximize efficiency and prevent damage during high winds.
- **Low-speed shaft**: This shaft connects the rotor to the gearbox. It rotates at the same speed as the rotor.
- **Gearbox**: This component increases the rotational speed from the low-speed shaft from the rotor to a higher speed suitable for the generator. It's essential because generators require a much faster rotation to produce electricity efficiently.
- **High-speed shaft**: After the gearbox increases the rotational speed, this shaft delivers the high-speed rotational energy to the generator.
- **Generator**: This device converts the mechanical energy from the rotor (transferred through the shafts) into electrical energy.

- **Anemometer**: This instrument measures wind speed and transmits wind speed data to the controller.
- **Wind Vane**: Measures wind direction and communicates with the yaw drive to orient the turbine properly concerning the wind.
- **Yaw drive**: This is used to turn the nacelle and the rotor towards or away from the wind to capture the most energy from the wind.
- **Yaw motor**: Powers the yaw drive.
- **Brake**: A mechanical, electrical, or hydraulic device to stop the rotor in emergencies.
- **Nacelle**: The enclosure that houses all of the generating components of a wind turbine, including the generator, gearbox, and yaw mechanisms.
- **Tower**: The structure that supports the nacelle and rotor. It raises them above obstacles and closer to stronger winds.
- **Controller**: The controller is responsible for starting up the WT at cut-in speed, regulating the power generation and rotor speed, minimizing the load fatigues, and shutting off the machine at cut-off wind speed.

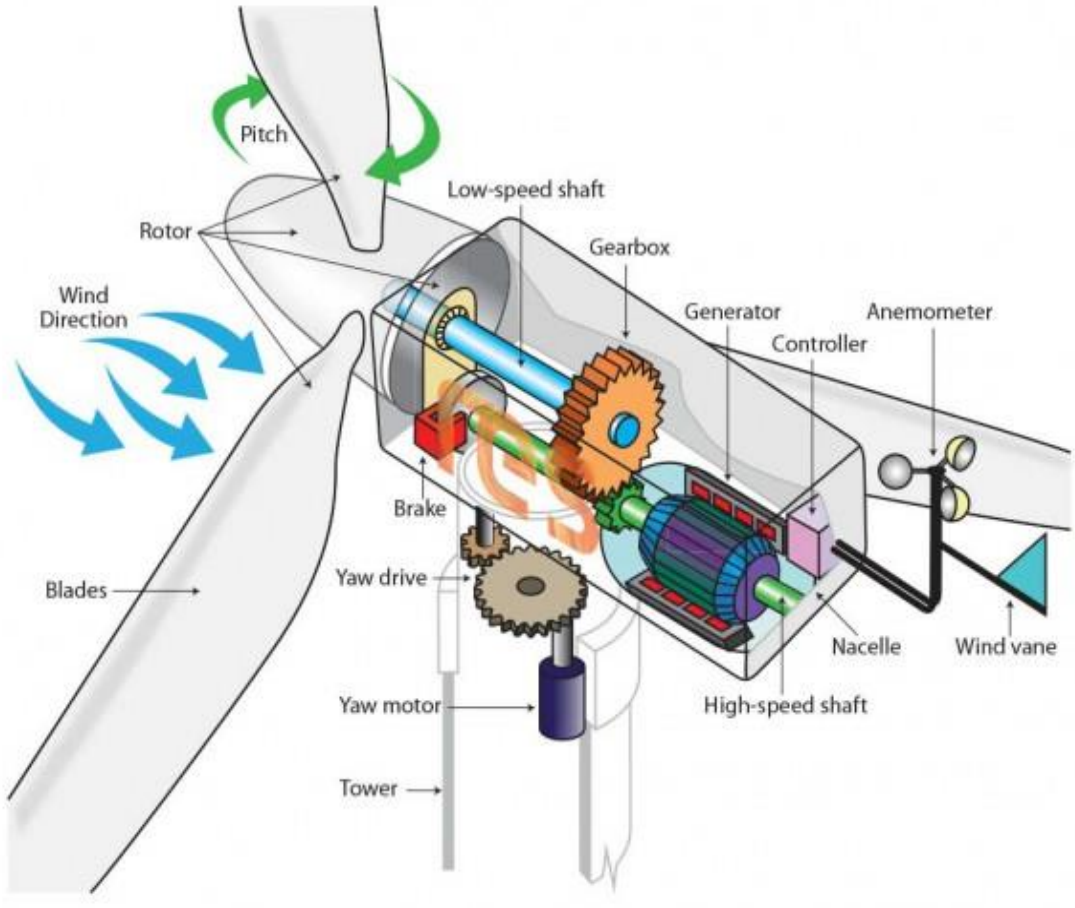


Figure 2.3 The HAWT main components. [66]

2.1.4. WECS operating regions

The WECS operating region is the range of wind speeds in which the system can operate safely and efficiently. The operating region can be divided into three main zones: the no-generation region (region 1), the partial-load region (region 2), and the

full-load region (region 3), as depicted in Figure 2.4. The boundaries of these zones depend on the characteristics of the wind turbine, the generator, and the control method. [67]

The no-generation region (region 1) is the zone where the wind speed is below the cut-in speed of the wind turbine. In this zone, the wind turbine does not rotate, generating no power. The cut-in speed is the minimum wind speed required to start the wind turbine. It depends on the aerodynamic design of the blades, the friction losses, and the generator characteristics [68].

The partial-load region (region 2) is the zone where the wind speed is between the cut-in speed and the rated speed of the wind turbine. In this zone, the wind turbine rotates and generates power, but the power output is less than the rated power of the system. The partial-load region is also called the maximum power point tracking (MPPT) region because the main objective of the control system in this zone is to extract the maximum possible power from the wind [68]. This can be achieved by adjusting the generator's rotational speed or the blades' pitch angle depending on the type of WECS [69].

The full-load region (region 3) is the zone where the wind speed is above the rated speed of the wind turbine. In this zone, the wind turbine rotates at maximum speed and generates its rated power. The full-load region is also called the power regulation region because the main objective of the control system in this zone is to limit the power output and protect the system from overloading. The main goal of control design in this region is to restrict the power captured by the WT to its rated power while maintaining the operation at the rated rotor speed at high wind speeds [3]. This goal can be achieved by a pitch control mechanism that effectively regulates the rotor loads and aerodynamic power without exceeding the constraints and limits of design [61]. The blade inertia determines the velocity of blade rotation and how robust the pitch mechanism is, so the maximum pitch rate of the WT model used is (8°/sec) [3], [6]. The pitch control consists of three types: individual pitch control (IPC) [7], cyclic pitch control, and collective pitch control (CPC).

This study focuses on CPC in region 3. The CPC is responsible for adjusting all blades simultaneously to the same pitch angle while achieving the rated power extraction and rated rotor speed from the WT. Figure 2.4 shows the power curve of the 5-MW wind turbine model used in this study.

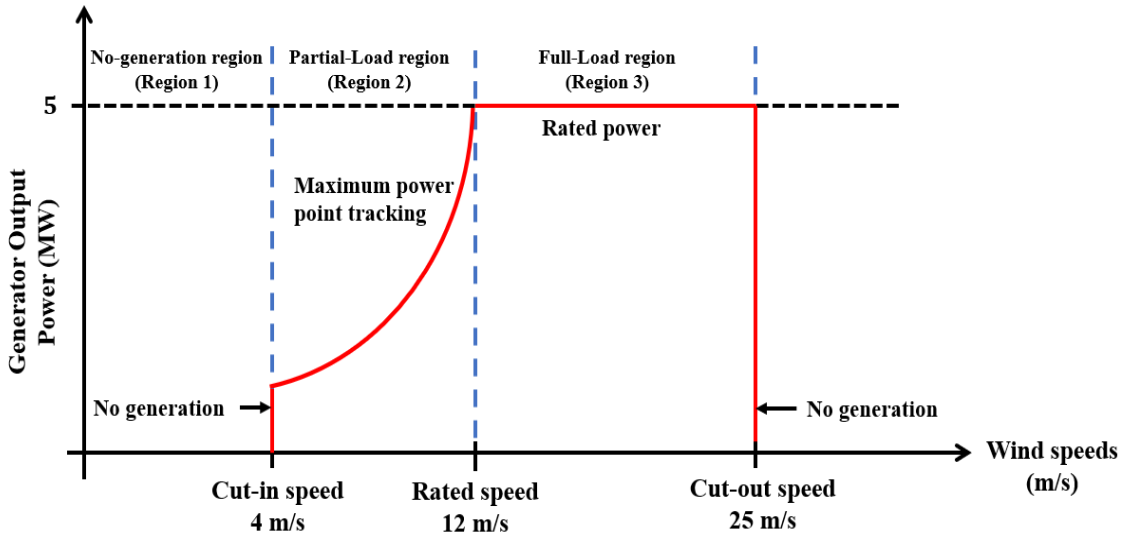


Figure 2.4 The wind turbine operational regions.

2.2. Wind Turbine System Analysis

2.2.1. Wind turbine system description and parameters

The WT model relies on Open-FAST (Fatigue, Aerodynamics, Structure, and Turbulence), an open-source software package [6]. Open-FAST is an aero-hydro-servo-elastic platform created by the NREL to model the dynamic reactions of both onshore and offshore wind turbines. Open-FAST allows users to turn on or off specific degrees of freedom (DOFs) to simplify or complicate the simulation. It offers three main models for testing and simulation purposes: the onshore, fixed-bottom offshore, and floating offshore. The performance of a 5-MW horizontal-axis onshore model and a floating offshore wind turbine model termed "OC3-Hywind spar buoy" [70] are tested in this study. The model framework, illustrated in Figure 2.5, shows the pitch angle controller, system dynamics, applied loads, external conditions, and their interactions for an offshore model. The onshore model has the same framework, excluding the hydrodynamic, platform, and mooring system dynamics. The model's main specifications are listed in Table 2.1. Additional information regarding the wind turbine modularization framework exists in [6], [70], [71].

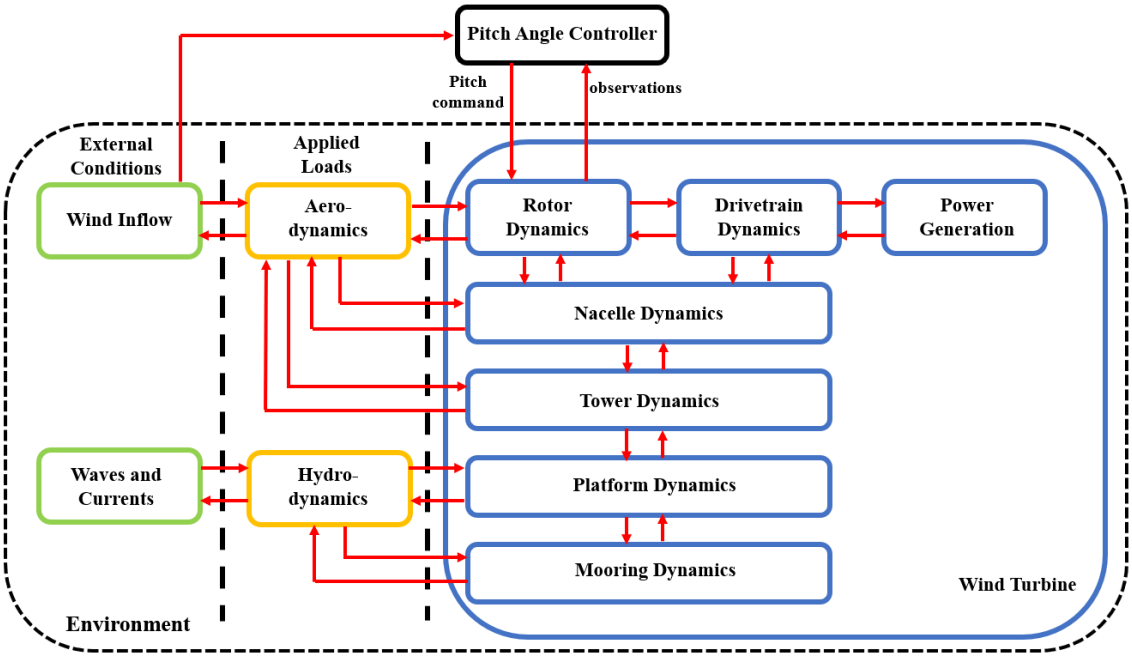


Figure 2.5 Modularization framework dynamics for offshore model [71]

Table 2.1 Wind turbine model mean specifications [6]

Parameter description	Value
Hub height	90 m
Rated generator speed	1173.7 rpm
Rated generator output power	5000 kW
(Cut in – rated - cut out) wind speeds	(4 – 11.4 – 25) m/s
Number of blades	3
Rotor diameter – Blade length	126 m – 63 m
Floating platform depth	120 m
Cut-in, Rated rotor speeds	6.9 rpm, 12.1 rpm
Pitch angle permissible range	$[0 - 90]\ degrees - \left[0 - \frac{\pi}{2}\right]\ rads$
Maximum pitch rate	± 8 degs/s (± 0.1396 rad/s)

2.2.2. The activated DOFs for simulation purposes

HAWTs' degrees of freedom (DOFs) are central to understanding how these structures respond to aerodynamic forces and moments caused by wind. These DOFs refer to the independent ways a turbine can move or deform. In HAWTs, these movements are critical to both the efficiency of energy capture and the structural

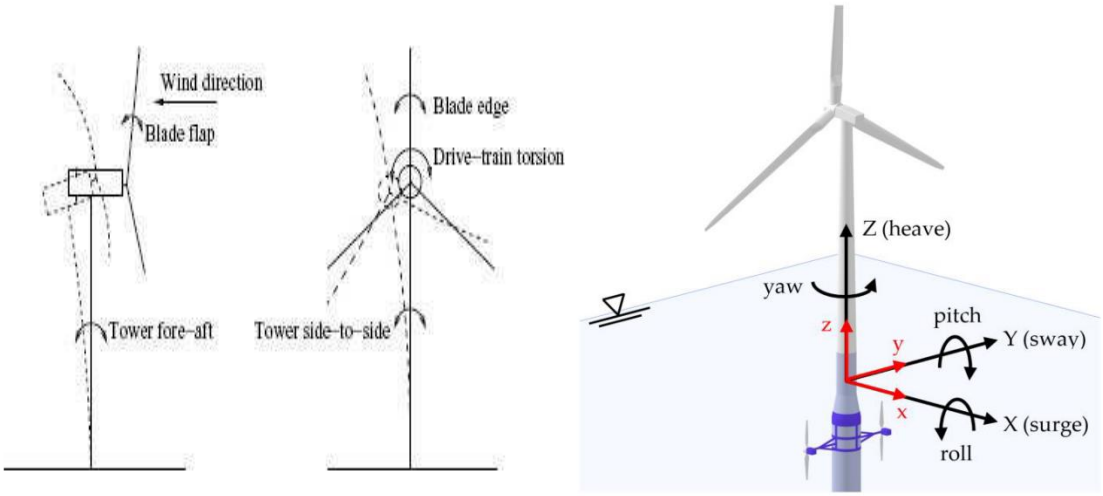
integrity of the turbine over time. Figure 2.6 and Table 2.2 represent these DOFs. In addition, a brief definition of the DOFs included in our WT model is introduced below [3]:

- **Flap-wise Modes:** These refer to the bending of the blades in the direction perpendicular to the rotor plane. The aerodynamic lift generated by the wind causes the blades to bend upwards, which can be considered the flapping motion of a bird's wings.
- **1st Edge-wise Mode:** This is the fundamental mode of edge-wise vibration and is often the most significant in blade dynamics. It involves the blade bending toward and away from the tower, as depicted in Figure (2.6-a), and it is critical to monitor and control this mode to prevent contact between the blade and the tower, especially in strong gusts of wind.
- **Rotational Flexibility:** This DOF involves twisting the blade along its length. Wind speed or direction changes can cause the blades to twist, affecting the angle of attack and the turbine's efficiency.
- **Variable Speed:** This is the turbine's ability to change the rotor's rotational speed in response to fluctuating wind conditions. The turbine can maximize energy capture and reduce mechanical stress by optimizing the rotational speed.
- **Yaw Bearing:** The yaw bearing allows the turbine to rotate around a vertical axis (yaw motion), enabling the rotor to align with the wind direction to capture maximum energy.
- **Side-to-Side Bending Modes:** This DOF refers to the side-to-side movement of the nacelle and the top of the tower, which can occur due to asymmetric loads on the rotor or turbulent wind conditions.
- **Fore-Aft Bending Mode:** This is the movement of the nacelle and the top of the tower back and forth in the direction of the wind. This motion is influenced by the wind pushing and pulling on the rotor.
- **Translation Modes:** These modes include the horizontal surge (movement along the wind direction), horizontal sway (movement perpendicular to the wind direction), and vertical heave (up and down movement). These translational movements can be due to wind shear, wave action (for offshore turbines), or seismic activity.
- **Rotational Modes:** These modes, as depicted in Figure (2.6-b) contain roll tilt (rotation about the longitudinal axis of the turbine), pitch tilt (rotation about the lateral axis), and yaw (rotation about the vertical axis). These rotational movements help the turbine adjust to changes in wind direction, distribute loads, and maintain stability.

The reduced-order WT model is where the drive train and the generator DOFs are only activated. This model is used for linearization to measure the disparity between WT models at different operating points. The design of the Gain-Scheduled PI (GSPI) controller, Fuzzy Model Predictive Controller (FMPC), and Linear-Quadratic-Regulator (LQR) represented in this thesis resides in this model. At the same time, the F-DDPG and CNN-C training resides in a medium-fidelity onshore environment. The activated dynamics during training are the blades, drive-train, generator, nacelle, and tower DOFs.

In the deployment phase, the controllers are evaluated in high-fidelity environments using both onshore and offshore wind turbine models. For the onshore

model, all DOFs are enabled except for the platform-related ones. In the offshore model, all DOFs are activated, including additional dynamics like platform DOFs, hydrodynamics, and mooring system dynamics, as detailed in Table 2.2 and shown in Figure 2.6. The controllers' abilities are tested to ensure performance generalization across different settings and operating conditions.



(a) (b)

Figure 2.6 The DOFs of the wind turbine model. (a) the tower and hub DOFs [72]**. (b) the platform DOFs** [73]

Table 2.2 OpenFAST HAWT 3-blade nonlinear model internal DOFs. [6]

Element	**Number of DOFs**	**Description**
Blades	2	Flap-wise modes per blade (3-blade) (1^{st} and 2^{nd})
	1	1^{st} Edge-wise mode per blade (3-blade)
Drive-Train	1	rotational flexibility
Generator	1	Variable speed
Nacelle	1	Yaw bearing
Tower	2	side-to-side bending modes
	2	fore-aft bending mode
Platform (for offshore model only)	3	Translation modes (horizontal surge, horizontal sway, vertical heave)
	3	Rotational modes (Roll tilt, pitch tilt, yaw)

2.2.3. OpenFAST/MATLAB/Simulink simulation tools

One of the critical features of OpenFAST is its ability to interface with MATLAB/Simulink, a widely used platform for model-based design and simulation across various engineering disciplines. This interaction enables the integration of OpenFAST's detailed wind turbine models with the extensive array of tools and functionalities available in Simulink. Users can design, simulate, and test advanced control strategies for wind turbines within Simulink's graphical environment. This allows for exploring novel control algorithms and their impact on turbine performance and longevity. In addition, the OpenFAST-Simulink interface facilitates the execution of dynamic simulations, enabling the assessment of wind turbine behavior under varying operational scenarios and environmental conditions. This helps identify potential issues, optimize turbine configurations, and enhance energy output.

The adhesive code of OpenFAST and its underlying modules are predominantly composed in Fortran, following the 2003 standard [6]. Additionally, modules can be developed in C or C++. The OpenFAST/Simulink framework resides on S-Function and interacts with module input files. The modules represent the wind inflow, aerodynamics, elastodynamics of blades and towers, and the servo dynamics of the controllers. Users could modify the input files according to the desired simulation purposes.

The controllers of the OpenFAST nonlinear WT mode are the torque, yaw, high-speed shaft brake, and pitch angle controller. Figure 2.7 represents the openFAST v2.4 WT model interface in Simulink. The main focus of this study is the design of the pitch angle controller.

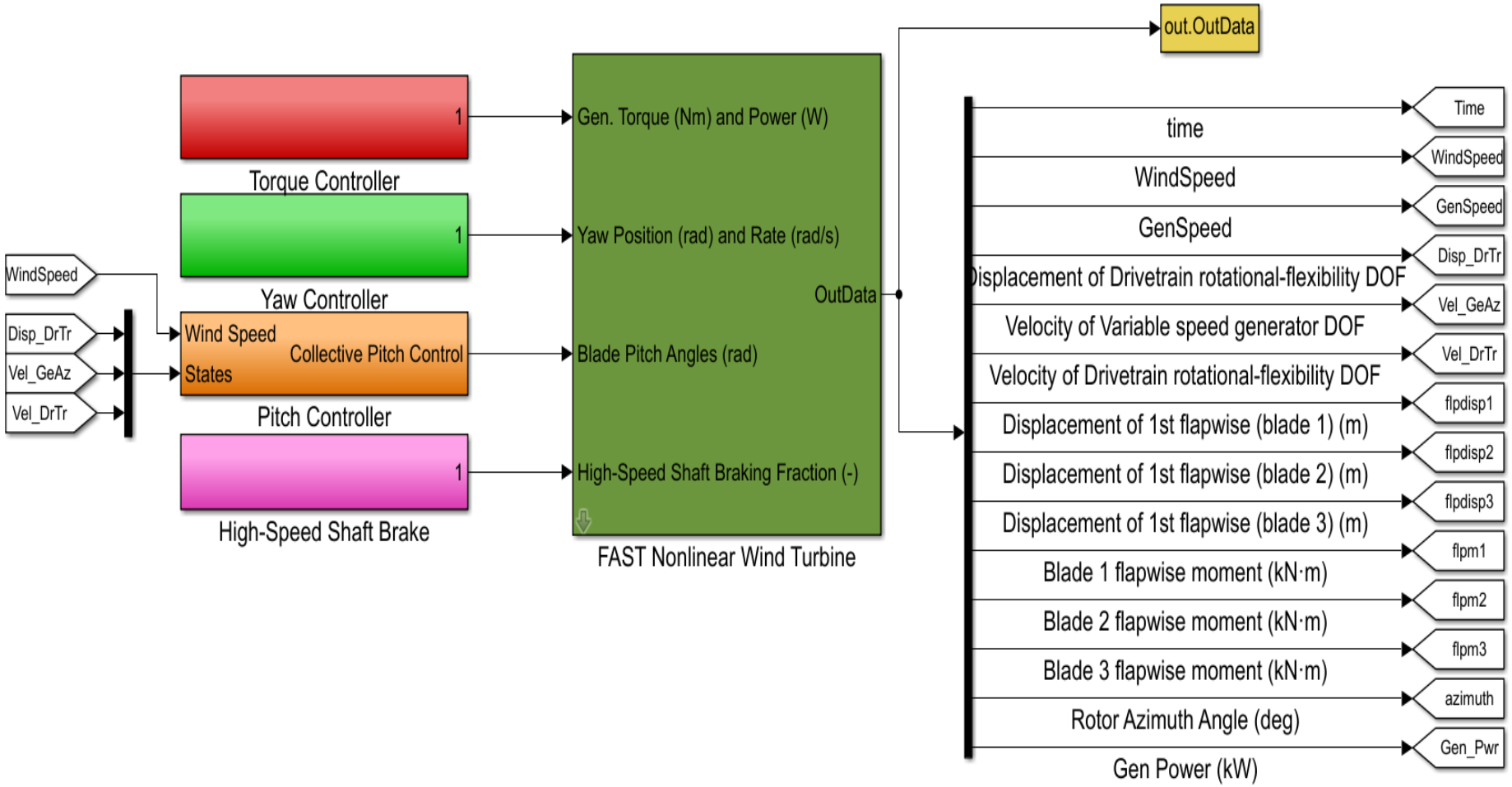


Figure 2.7 The OpenFAST/Simulink framework of a 5-MW wind turbine nonlinear model.

2.2.4. Disparity between wind turbine models

A WT model is time-varying due to uncertainties in wind inflow. To ensure the controller's robustness, measuring the disparity between the WT systems dynamics due to random perturbations of wind speeds is critical. Consecutively, a simplified analysis using the gap-metric criterion is introduced [8]. A reduced-order nonlinear WT model, whose generator and drive-train DOFs are only activated, is linearized using the Open-FAST package linearization tool at seven different operating points. The operating point is determined by the rotor speed, generator azimuth angle, and wind speed. The generator azimuth angle is defined as the angle between the rotor blades and a predefined reference orientation of these blades. The operational points are represented by mean wind speeds of (12, 14, 16, 18, 20, 22, and 24) m/s. The rotor speed is fixed at the rated value since the linearization is done in region three operation mode. At each mean wind speed, the Open-FAST linearization tool produces 36 linearized models at different generator azimuth angles with a 10-degree angle gap between each two consecutive models. The multi-blade coordinate transformation tool is used to calculate the average model of the 36 linearized models and express the integrated dynamics of turbine blades to a fixed (non-rotating) frame [74]. As a result, the only variable that distinguishes the average linearized models is wind speed. The average model at each operating point could be expressed in state-space form as in equations 2.1 and 2.2.

$$\dot{\boldsymbol{x}}_n = \boldsymbol{A}_{\boldsymbol{avg_n}}\boldsymbol{x_n} + \boldsymbol{B}_{\boldsymbol{avg_n}}u_{CPC} \tag{2.1}$$

$$y = \boldsymbol{C}_{\boldsymbol{avg_n}}\boldsymbol{x_n} \qquad n = 1, 2, \dots, 7 \tag{2.2}$$

The matrices $\boldsymbol{A}_{\boldsymbol{avg_n}}, \boldsymbol{B}_{\boldsymbol{avg_n}}$, and $\boldsymbol{C}_{\boldsymbol{avg_n}}$ contain constant parameters that describe the linearized model at a certain n^{th} operating point. The system states vector $\boldsymbol{x_n}$ which contains the following states: The torsional displacement of the drive train, the error in rotor speed, and the torsional speed of the drive train. u_{CPC}, y are collective pitch control input and generator speed output, respectively.

The operating points are selected based on the gap metric criterion, which evaluates the disparity of WT reduced-order linear model dynamics at these points. The dynamics of two linear models are considered different if the gap distance between the two models is more significant than a predetermined value. Equation (2.3) describes the maximum gap distance $D(G_n, G_m)$ between two single-input-single-output transfer functions (G_n, G_m) at average wind speeds (n, m). The gap distance values lie between [0, 1] where zero means no difference between models and one means quite different models. $\overline{G_n}$ and $\overline{G_m}$ are the conjugates of G_n and G_m. The correlation matrix representing the disparity and distances between the linearized models are shown in Table 2.3. Thus, the dynamics of the linearized models at a 2 m/s wind speed gap are considered sufficient to describe the WT system at all possible wind speeds in region 3.

$$D(G_n, G_m) = \left\| \frac{G_n - G_m}{\sqrt{1+G_n\overline{G_n}}\sqrt{1+G_m\overline{G_m}}} \right\|_\infty \tag{2.3}$$

Table 2.3 Gap metric correlation matrix of average linearized models

Transfer functions	G_{12}	G_{14}	G_{16}	G_{18}	G_{20}	G_{22}	G_{24}
G_{12}	0.000	0.149	0.244	0.324	0.381	0.430	0.512
G_{14}	0.149	0.000	0.099	0.185	0.246	0.300	0.381
G_{16}	0.244	0.099	0.000	0.088	0.152	0.208	0.263
G_{18}	0.324	0.185	0.088	0.000	0.065	0.123	0.194
G_{20}	0.381	0.246	0.152	0.065	0.000	0.059	0.103
G_{22}	0.430	0.300	0.208	0.123	0.059	0.000	0.072
G_{24}	0.512	0.381	0.263	0.194	0.103	0.072	0.000

2.3. Supervised Learning Pipeline

Supervised learning algorithms are one of the neural network training objectives, wherein a model learns to map input features to desired outputs based on labeled training data. This process unfolds through steps designed to optimize the network's parameters for the best possible performance on unseen data.

A supervised learning pipeline is a structured sequence of steps and processes used to develop, train, evaluate, and deploy machine learning models that rely on labeled data. In supervised learning, the algorithm learns from input-output pairs, where the input data (features) and the corresponding correct outputs (labels) are provided. The goal is to make accurate predictions on new, unseen data.

Initially, data preparation is essential, involving cleaning, normalization (like z-score normalization technique to achieve zero mean and one standard deviation), and potential augmentation to ensure the network receives high-quality and representative inputs. Subsequently, model architecture selection takes place, where the complexity of the neural network is specified based on the task. In our task, CNN is selected due to its powerful learning capability from time series data and it can capture the linear and non-linear relations between input states and desired output (CPC).

Once the network design is established, the weight initialization is performed, often using strategies like Glorot-Xavier Initialization [75], He initialization [76], or random numbers between [-1,1] to start training in a state that encourages convergent behavior. The core of the training is the iterative process where the model's predictions are compared against actual labels using a loss function, quantifying the error for each batch of data. The random numbers method is selected in our task because it gives efficient training progress.

The optimization algorithm adjusts the network's weights to minimize this loss. During the training of the deep neural networks, regularization techniques such as dropout or weight decay like Lasso (L1) and Ridge (L2) regularization techniques may be employed to prevent model overfitting by promoting the development of a more

generalized model. Simultaneously, techniques like batch normalization may be used to maintain stable distributions of activations throughout the network.

Furthermore, a validation process occurs in parallel to training, where a separate set of data not seen by the network during weight updates is used to monitor generalization and performance. This step is crucial for hyperparameter tuning (number of layers, number of neurons in each layer, learning rate, L2 regularization factor, the type of activation function, … etc) and for early stopping to halt training before overfitting occurs.

Finally, after training the model offline, the model undergoes evaluation using a testing set, providing an unbiased assessment of the network's performance on data that did not engage in the learning process. This phase ensures the robustness and generalizability of the trained neural network before it is deployed in applications. [77]

In our design of the proposed CNN-C, cleaned data are collected and then augmented based on the need. The collected data are not normalized since the normalization is to rescale the input features to have zero mean and one standard deviation to enhance the consistency of scalability. This normalization process requires previous data to obtain the mean and standard deviation values for scalability during the training. Since our task is deployed in real-time, this will require normalizing the input features at each iteration. This may be harsh at the beginning of the simulation due to the lack of previous data. The weight initialization is done using uniformly distributed random numbers between [-1 and 1]. Since our task requires a shallow CNN architecture, we are not afraid of overfitting problems, so regularization techniques are not implemented to simplify the design.

2.4. An Overview of RL-based Controllers

RL is a type of machine learning where an agent learns to make decisions by taking actions in an environment to achieve some goals [53]. Unlike supervised learning, where the model is trained on a pre-labeled dataset, RL involves learning through trial and error, using feedback from its own actions and experiences to improve over time. The fundamental objective of RL is to develop a policy that guides the agent in maximizing cumulative rewards over time.

In RL, the agent interacts with its environment at discrete time steps. At each time step, the agent observes the current state of the environment, decides on an action from a set of possible actions (in the case of a stochastic policy), and then takes that action. The environment responds to the action by presenting a new state and giving a reward, which can be positive or negative, reflecting the value of the action taken in that particular state. The agent's goal is to learn a policy — a mapping from states to actions — that maximizes the total sum of received rewards, often referred to as the return.

2.4.1. Fundamentals of Reinforcement Learning

In the context of our study on wind turbine control, the definition of the RL problem is integral to developing an effective control strategy. This definition comprises seven fundamental terminologies: agent, environment, states, reward

function, action, policy, and value function. Figure 2.8 shows the interaction between these seven components. Also, these elements are vital in shaping the learning process and the resulting controller's performance. By defining these components, we aim to construct a robust and efficient framework capable of addressing the unique challenges of WT operations' dynamic and complex nature.

- **Agent:** The agent is the controller or decision-maker interacting with and learning from the environment. The agent can be a robot, a computer program, a human, or an intelligent system. In our study, It is an intelligent computer algorithm.
- **Environment**: The environment is the framework that comprehensively contains external conditions, applied loads, and the dynamic behavior of the wind turbine, providing a realistic and challenging setting for the RL problem. In the context of this study, the term "medium-fidelity environment" represents the WT framework operating at a certain average wind speed in region 3. Meanwhile, the "high-fidelity environment" means WT framework operation at all wind speed ranges in region 3.
- **States**: It represents the current situation or condition of the wind-turbine dynamics. In this study, the WT system is characterized by low-dimensional states, which include the torsional displacement of the drive-train θ_{DT} , error in rotor speed $\Delta\omega_{rotor}$, and the torsional speed of the drive-train ω_{DT}. These states offer partial but informative insights into the WT's operational status.
- **Reward Function**: The reward function is the objective function that needs to be maximized throughout the training process. It is designed to ensure the generator's speed and power output remain close to optimal. The term "cumulative reward" is the sum of all the rewards that an agent receives during an episode. The cumulative reward is employed during the training and deployment as a performance evaluation metric.
- **Action**: It is the control command applied to the system. The actions can be deterministic or stochastic, discrete or continuous, finite or infinite. The action affects the state and the reward of the agent. In our RL framework, the collective pitch control signal is the continuous action taken by the agent. This action is critical as it adjusts the turbine blades' pitch angles, directly influencing the system's performance in region 3.
- **Policy**: It is the function or strategy that the agent follows to select actions in each state. The policy can be deterministic or stochastic, explicit or implicit, static or dynamic, off or on. The policy can be learned or given by an expert. In our study, we use a deterministic, implicit, dynamic, off-policy strategy that is learned through the agent’s interaction with the environment. The policy function is approximated by a neural network termed “actor network” that is trained to map states to actions.
- **Value function**: It is a function that estimates the long-term value or desirability of each state or state-action pair. The value function can be state-value or action-value, deterministic or stochastic, scalar or vector. The value function can be learned or derived from the policy or the reward. In this study, the value function is the expected discounted sum of all the rewards. The agent's goal in RL is to learn a policy that maximizes the expected cumulative reward over many episodes. In addition, the value function is a state-value, deterministic, scalar function approximated using

a neural network called a “critic network.” The critic network is trained to minimize the difference between the actual and the expected cumulative reward.

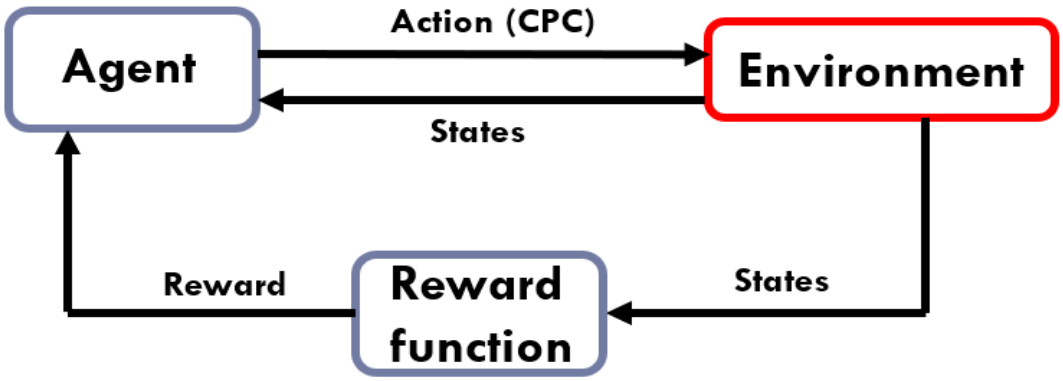


Figure 2.8 The interaction between different components of the RL problem

2.4.2. Dynamic Programming and Bellman Equation

Dynamic programming (DP) is a set of methods that may be utilized to calculate optimum policies when provided with a flawless environment model in the form of a Markov decision process (MDP). Classical dynamic programming techniques have little practical use in reinforcement learning due to their reliance on a flawless model and significant computing cost. However, they remain crucial from a theoretical standpoint [53].

Dynamic programming is a mathematical optimization method that solves complex problems by breaking them into simpler subproblems. It relies on the principle of optimality, which asserts that the optimal solution to a problem incorporates optimal solutions to its subproblems. This method is particularly effective in cases where subproblems overlap, and the problem exhibits a recursive structure, allowing solutions to be built up from solving smaller instances of the same problem [53]

The relationship between dynamic programming and the Bellman equation is foundational in the context of reinforcement learning and decision processes. The Bellman equation, named after Richard Bellman, provides a recursive decomposition for the value function of a decision problem. It expresses the principle of optimality in a mathematical form, stating that the value of a state under an optimal policy is equal to the expected return from the best action taken in that state, considering both immediate rewards and the discounted value of subsequent states. The details of the bellman’s optimality are represented in Subsection 2.4.3.

In RL and control theory, the Bellman equation finds optimal policies and value functions for deterministic and stochastic decision problems. Dynamic programming applies the Bellman equation through algorithms like value iteration and policy iteration [57]. These algorithms iteratively update value estimates for all states (or state-action pairs) based on the Bellman equation until they converge to the optimal solution. Therefore, dynamic programming is a practical framework for implementing the

recursive strategy implied by the Bellman equation in discrete and continuous decision-making problems.

2.4.3. The role of RL in optimal control problems

RL addresses the issue of finding the best control strategy in a discrete-time setting. This is commonly represented as a Markov decision process (MDP) [78]. Because of its roots in artificial intelligence, this formalism exhibits some distinct characteristics in its terminology and particular technical decisions, such as utilizing maximization and discounting. Thus, the MDP applicability in control theory is explained in this Subsection.

For a model-based continuous-state-space RL problem, a (MDP) is composed of a system's state space S, the action space A for inputs, the system's transition function T representing the dynamics, and a reward function R that represents the negative of costs. The state transitions inherently possess a stochastic nature. Therefore, the system's state s_k at a given time k transitions to a new state s_{k+1} in a probabilistic manner following action a_k. This new state is sampled from the probability density transition function $T(s_k, a_k, \cdot)$, which has to establish a legitimate probability density. The dot in $T(s_k, a_k, \cdot)$ represents the following unknown probabilistic states (probability distribution of being in the following states). Thus, the transition function T aggregates such probability densities, mapping across $S_k \times A_k \times S_{k+1} \rightarrow [0, \infty)$.

For discrete-state-space RL problem, the new state is sampled from the probability mass transition function. At this case, the transition function gives the actual probability of transitioning to a specific state s_{k+1}, as shown in (2.4). Thus, the transition function T maps across $S_k \times A_k \times S_{k+1} \rightarrow [0, 1]$.

$$\boldsymbol{T(s_k, a_k, s_{k+1}) = \mathbf{Pr}(s_{k+1} | s_k,\ a_k)} \tag{2.4}$$

Where (Pr) means the probability of being in a state s_{k+1} based on the current state s_k and action a_k.

As a consequence of the new state s_{k+1}, the reward value $r_{k+1} = R(s_k, a_k, s_{k+1})$ is calculated, where $R : S_k \times A_k \times S_{k+1} \rightarrow \mathbb{R}$. The reward assesses the immediate impact of action when moving from state s_k to the state s_{k+1}, but generally does not provide information about its long-term consequences.

The agent (controller) selects actions based on its policy μ. The policy is deterministic in this case $\mu : S \rightarrow A$, where $a_k = \mu(s_k)$. According to the following state-action trajectory pairs $((s_0, a_0), (s_1, a_1), (s_2, a_2), \dots, etc)$ and their related rewards $(r_1, r_2, r_3, \dots, etc)$, the cumulative discounted return over an infinite horizon is calculated as follows:

$$\boldsymbol{\sum_{k=0}^{\infty} \gamma^k r_{k+1}} \tag{2.5}$$

The discount factor $\gamma \in (0,1]$. The evaluation of a policy μ from s_0 is the estimated return considering the random transitions acquired from adhering to the policy μ (assuming the initial action a_0 following the policy):

$$\boldsymbol{V^{\mu}(s) = \mathbb{E}_{s' \sim T(s, \mu(s), \cdot)} \left\{ \sum_{k=0}^{\infty} \gamma^k R(s, \mu(s), s') | s_0 = s, \mu \right\}} \tag{2.6}$$

Where s' is the quantities of the next states. $\mathbb{E}_{s'\sim T(s,\mu(s),\cdot)}$ notation refers to the expectation of a discounted future reward according to the next estimated state being in the probability distribution of the transition function.

Function $V(s)$ is termed as V-function or state-value function. The V-function is the measure for how valuable it is to be in a certain state. The control goal is to attain the optimal policy μ^* that achieves the optimal (maximum) V-function:

$$V^*(s) \coloneqq \max_{\mu} V^{\mu}(s_0) = \max_{\mu} \mathbb{E}_{s'\sim T(s,\mu(s),\cdot)} \left\{ \sum_{k=0}^{\infty} \gamma^k R(s,\mu(s),s') | s_0 = s, \mu \right\}, \forall\, s_0 = s \tag{2.7}$$

To express this expression recursively, break equation (2.7) to r_1 and the rest of the subsequence rewards (bellman's optimality equation). This forms equations (2.8) and (2.9):

$$\boldsymbol{V^*(s) = \max_{\mu} \mathbb{E}_{s'\sim T(s,\mu(s),\cdot)} \{r_1 + \sum_{k=1}^{\infty} \gamma^k R(s,\mu(s),s') | s_0 = s, \mu\}, \forall\, s_0} \tag{2.8}$$

$$\boldsymbol{V^*(s) = \max_{\mu} \mathbb{E}_{s'\sim T(s,\mu(s),\cdot)} \{r_1 + \gamma V^{\mu}(s') | s_0 = s, \mu\}, \forall\, s_0} \tag{2.9}$$

It should be noted that any policy that achieves the highest values in the previous equation is considered optimal. MDPs have a notable characteristic where, given the right circumstances, there is a deterministic optimum policy that maximizes the value, even if the transitions are stochastic. Hence, the goal is to optimize the predicted discounted return over an unlimited time horizon. This is tedious, especially in tasks with many expectations (e.g., the number of movements(states) the player should make to win a chess game increases exponentially at each step up. According to one estimate, there are over 9 million variations after just three moves each, 288 billion different possible positions after four moves each, and 318,000,000,000 ways to play just the first four moves [79]. Thus, dynamic programming is used to discretize the optimality problem to suboptimal problems, which could be more accessible to solve recursively with fewer steps to look ahead. Thus, the value-iteration and the policy-iteration algorithms are considered the earliest model-based dynamic-programming RL algorithm.

In control theory, problems are often solved without discounting (using a discount factor of $\gamma = 1$), and with the potential for rewards to be unbounded. This guarantees that the solutions (value functions) remain finite and the problem remains adequately defined. Also, it is necessary to apply specific stability conditions. On the other hand, in RL, the discount factor is typically set to a value less than one, and rewards are considered to be bounded. This approach inherently ensures that value functions are manageable and avoids issues related to stability. The challenge of theoretically integrating the goal-driven approach of AI-based RL, which prioritizes performance, with the stability perspective of traditional control theory remains unresolved. [57]

In the context of RL, Q-functions $Q: S \times A \to \mathbb{R}$ are used instead of V-functions. The difference between them is that value functions represent the expected return of following a policy from a given state. In contrast, Q-functions represent the expected return of taking a specific action from a given state and following a policy. Here is a simple example to illustrate the difference. Suppose you are in a maze with four

possible actions: up, down, left, and right. The value function tells you how good it is to be in each maze cell, assuming you follow some policy (e.g., always go right). The Q-function tells you how good it is to take each action from each maze cell, assuming you follow the same policy afterward. The equation representing Q function is described below:

$$\boldsymbol{Q^{\mu}(s,a) = \mathbb{E}_{s' \sim T(s,a,\cdot)} \{R(s,a,s') + \gamma V^{\mu}(s') | s_0 = s, a_0 = a, \mu \, from \, a_1\}} \quad (2.10)$$

The optimal Q-function Q^* is determined using V^* in equation (2.9). The rationale for favoring Q-functions is straightforward: once Q^* is attainable, an optimum policy μ^* may be calculated by choosing the action with the highest ideal Q-value in each state:

$$\boldsymbol{\mu^*(s) = \operatorname*{argmax}_a Q^*(s,a)} \quad (2.11)$$

If numerous actions maximize the return, any of them might be considered optimum. Conversely, the equation to calculate μ^* from V^* is more intricate and, importantly, relies on a model that is typically not accessible in RL. A policy μ is considered greedy in Q if it fulfills $\mu^*(s) \in \operatorname*{argmax}_a Q^*(s,a)$ for any Q-function. To discover an optimum policy, one must first determine the optimal Q-function and then derive a greedy policy based on this function. It is essential to observe that the state value functions may be readily related to Q-functions, $V^{\mu}(s) = Q^{\mu}(s, \mu(s))$ and $V^*(s) = \max_a Q^*(s,a) = Q^*\big(s, \mu^*(s)\big)$.

The Q-functions Q^{μ} and Q^* are defined recursively through the Bellman equations, which are derived from the definitions of the Q-functions. These equations play a crucial role in reinforcement learning algorithms. The Bellman equation characterizing Q^{μ} asserts that the value of executing action a while being in state s and considering the policy μ equals the estimated sum of instant observed reward and the discounted value attained by μ in the subsequent state s' :

$$Q^{\mu}(s,a) = \mathbb{E}_{s' \sim T(s,a,\cdot)} \{R(s,a,s') + \gamma Q^{\mu}(s', \mu(s')) | s_0 = s, a_0 = a, \mu \, from \, a_1\} \quad (2.12)$$

The optimality principle of the Bellman equation characterizes Q^* and asserts that the optimal Q-value of executing action a while being in state s equals the estimated sum of the instant reward and the discounted optimal Q-value obtained by the optimal action in the subsequent state:

$$\boldsymbol{Q^*(s,a) = \mathbb{E}_{s' \sim T(s,a,\cdot)} \left\{ R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \, | s_0 = s, a_0 = a \right\}} \quad (2.13)$$

Since control-theoretic formulations of optimum control are often deterministic, it is essential to look at the deterministic particular case of the framework. For deterministic cases, the transition density function $T(s, a, \cdot)$ allocates every probability mass to a distinct future state s', resulting in deterministic dynamics $s' = \bar{T}(s,a)$, where $\bar{T} : S \times A \to S$. In addition, the reward function can be simplified to the following: $\bar{R}(s,a) = R\big(s,a,\bar{T}(s,a)\big)$, for $\bar{R} : S \times A \to \mathbb{R}$. Also, in the deterministic

cases, the expectation notations in the value and Q functions are removed to be a unique, deterministic trajectory with a cumulative reward in the following form:

$$\boldsymbol{V^{\mu}(s) = \sum_{k=0}^{\infty} \gamma^{k} \bar{R}(s,a) | s_0 = s, \mu} \quad (2.14)$$

$$\boldsymbol{Q^{\mu}(s,a) = \bar{R}(s,a) + \gamma V^{\mu}(s') | s_0 = s, a_0 = a, \mu\ from\ a_1} \quad (2.15)$$

The Bellman optimality equation at the deterministic case becomes:

$$\boldsymbol{Q^{*}(s,a) = \bar{R}(s,a) + \gamma \max_{a'} Q^{*}(\bar{T}(s,a), a') | s_0 = s, a_0 = a} \quad (2.16)$$

For additional details regarding the derivation of the Riccati equation from the Bellman optimality equation, [80] and [81] explain the concept of what is called the Hamilton-Jacobian-Bellman (HJB) equation and its relation to the Riccati equation in the context of dynamic programming and RL.

2.4.4. Stochastic and deterministic RL policies

a) Stochastic RL policy

A stochastic policy μ_s is a policy that maps each state to a probability distribution over actions. A probability distribution is a function that allocates a likelihood to every possible event (here, the events refer to actions taken in given states), ensuring that the total of these probabilities adds up to 1. Given a state (s), the agent will choose an action (a) randomly based on the probability distribution. This policy is represented by a function $\mu_s(A|S)$, where S is the system state space, A is the action space, and $\mu_s(a|s)$ is the probability of taking action a in state s, where $\mu_s : S \times A \rightarrow [0,1]$. For each state $s \in S$, the function $\mu_s(s, \cdot)$ gives a probability distribution over the action space A, such that: $\sum_{a \in A} \mu_s(s,a) = 1$. The advantage of a stochastic policy is that it can capture the uncertainty in the environment. [53]

Consider a simple numerical example to illustrate the idea of iterative learning and dynamic programming of the stochastic policy. The following example is a simple finite-discrete-space environment with two states, $s^{(1)}$ and $s^{(2)}$, and two actions, $a^{(1)}$ and $a^{(2)}$. Initially, the stochastic policy randomly assigns the probabilities to each action in each state. These initial probabilities can be set arbitrarily to encourage exploration. In some cases, they may start out as equal probabilities if no prior knowledge favors one action over another. The process is calculated iteratively through dynamic programming.

Assume the initial value $V_k^{\mu}(s)$ of being at certain state $s = s_0$ is expressed in Table 2.4:

Table 2.4 Value of being in a particular state

State	$V_k^{\mu}(s)$
$s^{(1)}$	+1
$s^{(2)}$	+1

Assume the following initial guess of the probabilities assigned by the stochastic policy μ_s are all the same, as expressed in Table 2.5:

Table 2.5 Stochastic policy probabilities based on state-action pair

State	Action	$\mu_s(a\|s)$
$s^{(1)}$	$a^{(1)}$	0.5
	$a^{(2)}$	0.5
$s^{(2)}$	$a^{(1)}$	0.5
	$a^{(2)}$	0.5

This means that being in state $s^{(1)}$, the policy will choose action a_1 and action $a_2$with a 50% chance. Being in state s_2, the policy will choose action a_1 and action $a_2$with a 50% chance.

The probabilities $\Pr(s'|s,a)$ of moving from the current state s to the next state s' based on the current action a are calculated (they are assumed in this illustrative example) using what is called by the transition function $T(s,a,s')$ which represent the state dynamics of the system. The reward acquired through transition from state s to the next state s' is $R(s,a,s')$. Table 2.6 illustrates the quantities.

Table 2.6 Transition function probabilities and reward acquired

State	Action	Next state	$\Pr(s'\|s,a)$	$R(s,a,s')$
$s^{(1)}$	$a^{(1)}$	$s^{(1)}$	0.8	+10
		$s^{(2)}$	0.2	+5
$s^{(1)}$	$a^{(2)}$	$s^{(1)}$	0.3	+7
		$s^{(2)}$	0.7	+14
$s^{(2)}$	$a^{(1)}$	$s^{(1)}$	0.6	+20
		$s^{(2)}$	0.4	+4
$s^{(2)}$	$a^{(2)}$	$s^{(1)}$	0.9	+10
		$s^{(2)}$	0.1	+8

The next iteration of the Q-function is calculated based on the equation (2.19):

$$\boldsymbol{Q^{\mu}_{k+1}(s,a)} = \mathbb{E}_{s'\sim T(s,\mu(s),\cdot)}\left\{\boldsymbol{r_1+\gamma V^{\mu}_k(s')|s_0=s,a_0=a,\mu}\right\},\forall\, \boldsymbol{s_0} = \sum_{s',r}\boldsymbol{\Pr(s'|s,a)(R(s,a,s')+\gamma V^{\mu}_k(s'))} \quad (2.17)$$

The next iteration of the V-function is calculated as described in equation (2.20):

$$\boldsymbol{V^{\mu}_{k+1}(s)} = \mathbb{E}_{s'\sim T(s,\mu(s),\cdot)}\left\{\boldsymbol{r_1+\gamma V^{\mu}_k(s')|s_0=s,\mu}\right\},\forall\, \boldsymbol{s_0} = \sum_{a}\boldsymbol{\mu_s(a|s)}\sum_{s',r}\boldsymbol{\Pr(s'|s,a)(R(s,a,s')+\gamma V^{\mu}_k(s'))} \quad (2.18)$$

According to Table 2.5 and Table 2.6, let's consider taking action $a^{(1)}$ from state $s^{(1)}$ and the transition to the next state is $s^{(1)}$ or $s^{(2)}$. The Q-function will be calculated as follows (assuming a discount factor of $\gamma = 0.9$):

$$\boldsymbol{Q^{\mu}_{k+1}(s,a)} = \sum_{s',r}\boldsymbol{\Pr(s'|s,a)(R(s,a,s')+\gamma V^{\mu}_k(s'))} \quad (2.19)$$

$$\boldsymbol{Q_{k+1}^{\mu}(s^{(1)},a^{(1)}) = \Pr(s^{(1)}|s^{(1)},a^{(1)})\left(R(s^{(1)},a^{(1)},s^{(1)}) + \gamma V_k^{\mu}(s^{(1)})\right) + \Pr(s^{(2)}|s^{(1)},a^{(1)})\left(R(s^{(1)},a^{(1)},s^{(2)}) + \gamma V_k^{\mu}(s^{(2)})\right)} \quad (2.20)$$

$$\boldsymbol{Q_{k+1}^{\mu}(s^{(1)},a^{(1)}) = 0.8(10 + 0.9 * 1) + 0.2(5 + 0.9 * 1) = 9.9} \quad (2.21)$$

let's consider now taking action $a^{(2)}$ from state $s^{(1)}$ and the transition to the next state is $s^{(1)}$ or $s^{(2)}$:

$$\boldsymbol{Q_{k+1}^{\mu}(s^{(1)},a^{(2)}) = \Pr(s^{(1)}|s^{(1)},a^{(2)})\left(R(s^{(1)},a^{(2)},s^{(1)}) + \gamma V_k^{\mu}(s^{(1)})\right) + \Pr(s^{(2)}|s^{(1)},a^{(2)})\left(R(s^{(1)},a^{(2)},s^{(2)}) + \gamma V_k^{\mu}(s^{(2)})\right)} \quad (2.22)$$

$$\boldsymbol{Q_{k+1}^{\mu}(s^{(1)},a^{(2)}) = 0.3(7 + 0.9 * 1) + 0.7(14 + 0.9 * 1) = 12.8} \quad (2.23)$$

While the V-function is calculated as following:

$$\boldsymbol{V_{k+1}^{\mu}(s^{(1)}) = \mu_s(a^{(1)}|s^{(1)})\left(Q_{k+1}^{\mu}(s^{(1)},a^{(1)})\right) + \mu_s(a^{(2)}|s^{(1)})\left(Q_{k+1}^{\mu}(s^{(1)},a^{(2)})\right)} \quad (2.24)$$

$$\boldsymbol{V_{k+1}^{\mu}(s^{(1)}) = 0.5(9.9) + 0.5(12.8) = 11.35} \quad (2.25)$$

The rest of the value functions at each case are calculated the same way to result in the Table 2.7:

Table 2.7 state-value function based on Q-function

State	Action	Next state	$\Pr(s'\|s,a)$	$R(s,a,s')$	$Q_{k+1}^{\mu}(s,a)$	$V_{k+1}^{\mu}(s)$
$s^{(1)}$	$a^{(1)}$	$s^{(1)}$	0.8	+10	9.9	11.35
		$s^{(2)}$	0.2	+5		
$s^{(1)}$	$a^{(2)}$	$s^{(1)}$	0.3	+7	12.8	
		$s^{(2)}$	0.7	+14		
$s^{(2)}$	$a^{(1)}$	$s^{(1)}$	0.6	+20	14.5	12.6
		$s^{(2)}$	0.4	+4		
$s^{(2)}$	$a^{(2)}$	$s^{(1)}$	0.9	+10	10.7	
		$s^{(2)}$	0.1	+8		

Based on the previously calculated Q-functions, the stochastic policy μ_s is updated according to predetermined criteria (assumed to be known and not described) to give a higher choice probability to the action that maximizes the Q function for the next iteration:

$$\boldsymbol{\mu^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)} \quad (2.26)$$

Now, Table 2.5 is updated to Table 2.8:

Table 2.8 Updated stochastic policy probabilities

State	Action	Probability
$s^{(1)}$	$a^{(1)}$	0.2
	$a^{(2)}$	0.8
$s^{(2)}$	$a^{(1)}$	0.7
	$a^{(2)}$	0.3

This numerical example illustrates how to deal with the notations and gives a solid background on the bellman optimality principle.

The probabilities in a stochastic policy within the context of RL are typically determined based on the following factors:

- **Initial Policy (Exploration):** When an agent begins learning in an environment, it may start with a random policy or some initial heuristic. These initial probabilities can be set arbitrarily to encourage exploration. In some cases, they may start out as equal probabilities if no prior knowledge favors one action over another.
- **Learning Algorithm (Exploitation):** As the agent interacts with the environment, it collects data on the rewards received for taking certain actions in specific states. RL algorithms update the policy based on this data. Actions that have historically led to higher returns will gradually be assigned higher probabilities, while less beneficial actions will receive lower probabilities.
- **Policy Gradient Methods:** These methods parameterize the policy and optimize the parameters directly through gradient ascent on the expected return. The probabilities are functions of the policy parameters updated to increase the likelihood of actions that lead to better outcomes.
- **Value Function Methods:** These methods first estimate the value of each action in a state and then derive a policy based on these values. A common approach is to use a softmax function over action values to create a probabilistic policy. Actions with higher estimated values have higher probabilities.
- **Exploration vs. Exploitation Trade-off:** To balance the need to explore new actions with the need to exploit known rewarding actions, the probabilities might be adjusted to ensure some level of randomness. Algorithms like epsilon-greedy, softmax exploration, or adding noise to policy parameters are used to handle this trade-off.
- **Policy Improvement:** In methods like policy iteration, the policy is improved by making it greedy concerning the value function. This can make the probabilities more deterministic over time as the agent becomes more confident in the optimal actions.

b) Deterministic RL policy

A deterministic policy is a policy where, for each state, one specific action is always taken. No randomness is involved in the decision-making process; given a state, the action is determined with certainty. The deterministic policy $\mu_d(s)$ is the function that maps the environment states (s) to a specific action (a), i.e. $\mu_d : S \rightarrow A$, the subscript d in μ_d refers to deterministic policy, S is the state space, and A is the action space. The action is exerted deterministically (i.e., the probability of taking the action is 1). This inherently means that the Q-function is the same as the V-function in this case, unlike the stochastic policy.

Considering the example illustrated in the stochastic policy Subsection (2.4.4-a), the deterministic policy will always choose the same action being in the same state (i.e., always selecting action $a^{(2)}$ when being in a state $s^{(1)}$). By doing so, the achieved value will be the same even though the action taken does not necessarily maximize the value function. So, adding exploration noise to the action is mandatory to explore whether other actions could lead to better value.

In continuous control problems like WT cases with infinite action space, using stochastic policies based on the probabilistic manner achieves high computational time. There are several reasons behind this. First, the quantization is needed for the continuous action space. Second, this quantization could miss some actions that achieve global optimality. Third, as the number of quantized actions increases, the need to calculate all the necessary probabilities to maximize the value function also increases. Although the stochastic policies are better for the uncertainties in the WT model, the deterministic policy is the suitable choice from the computational and learning point of view. Thus, the deterministic policy is used in this thesis.

The differences between the deterministic and stochastic policies could be summarized in the following key points:

- **Certainty vs. Uncertainty**: In a deterministic policy, each state's action is certain and fixed. In a stochastic policy, there is uncertainty, and multiple actions can be taken from the same state with different probabilities.
- **Exploration**: Stochastic policies inherently include exploration, as there is a non-zero probability of choosing less-favored actions. Deterministic policies require an external mechanism to incorporate exploration. $\epsilon -$greedy method is used in the case of discrete-action-space problems, while the Gaussian noise or Ornstein-Uhlenbeck noise methods are used in the continuous-action-space problems.
- **Complexity**: Deterministic policies are simpler and can be more straightforward to implement. However, they may not capture the complexity of particular environments where a degree of randomness in action selection can be advantageous.
- **Convergence**: In some cases, especially in environments with noise and uncertainty, stochastic policies can lead to better convergence properties during learning than deterministic policies.

2.4.5. Q-iteration algorithm

Considering the example illustrated in the stochastic RL policy Subsection (2.4.4-a), the process is repeated until convergence occurs. This example illustrates the concept of a Q-iteration algorithm [57]. The algorithm relies on the idea of implicitly choosing the optimal policy that maximizes the Q-function as described in following steps described in algorithm 1:

Algorithm 1: Q-iteration

Input: $T, R, \gamma, threshold\ c$

1. Initialize Q −function $Q_0(s,a) = 0, \forall\ s, a$
2. **For** k = 1, 2, … **do**
   a. **For** every state-action pair (s,a) **do**

$$\boldsymbol{Q_{k+1}}(\boldsymbol{s},\boldsymbol{a}) = \sum_{\boldsymbol{s'},\boldsymbol{r}} \mathbf{Pr}(\boldsymbol{s'}|\boldsymbol{s},\boldsymbol{a}) \left(\boldsymbol{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s'}) + \boldsymbol{\gamma} \max_{\boldsymbol{a'}} \boldsymbol{Q_k}(\boldsymbol{s'},\boldsymbol{a'})\right) \quad (2.27)$$

   b. **End for**
3. **Until** $|Q_{k+1} - Q_k|_\infty \leq\ c$

Output: $\hat{Q}^* = Q_{k+1}$, $\hat{\mu}^*$ greedy in $\hat{Q}^*$

2.4.6. Policy iteration algorithm

As mentioned before, the Q-iteration focuses on computing the Q-value for all possible state-action pairs. The optimal policy is implicitly chosen based on the optimal Q-value in a certain state. Unlike this method, the policy iteration algorithm is an iterative process combining two key points: policy evaluation and policy improvement. The policy evaluation step is based on keeping the current policy fixed (even if it is not optimal) and compute the value function for that policy under all states. The policy is then improved by finding the best action that maximizes via a step lookahead. The mathematics background of the policy iteration algorithm is described in detail in algorithm 2.

A simplified example to illustrate the idea is explained. Consider a simple 3x3 grid world environment as a numerical example to illustrate the differences between policy iteration and Q-iteration. In this environment, an agent can move in four directions: up, down, left, and right. The goal is to reach a specific location on the grid.

$s^{(1)}$	$s^{(2)}$	$s^{(3)}$
$s^{(4)}$	$s^{(5)}$	$s^{(6)}$
$s^{(7)}$	$s^{(8)}$	$s^{(9)}$

- **States**: The agent can be in any cell of a 3x3 grid (9 states in total).
- **Actions**: Up (U), Down (D), Left (L), Right (R).
- **Rewards**: Moving into the goal state gives +10, hitting a wall (moving off the grid) gives -1, and any other movement gives -1.

Assume the goal is at the bottom-right corner (state 9). The initial policy and value functions are arbitrary. Assume the initial policy (chosen arbitrarily) always moves right, and all states have zero initial value function.

The policy iteration algorithm will behave as follows:

1. Policy evaluation step:
    a. Calculate the value of each state under the "always move right" policy.
    b. For most states, this results in -1 (for moving) plus the value of the state to the right (initially 0).
    c. This continues until the values converge under this policy, reflecting the cost of reaching the goal state from any state by moving right.
2. Policy Improvement:
    a. For each state, check if changing the action (to U, D, L, or R) would improve the value.
    b. After evaluation, we might find that in the second row, moving down towards the goal is better than moving right.
    c. Update the policy accordingly and repeat the evaluation.

The Q-iteration algorithm will behave as follows:

a. Iteratively update the value of each state based on the Bellman optimality equation.
b. After the first iteration, the value of states adjacent to the goal increases because moving into the goal yields +10.
c. This update propagates back through the grid until the values converge, indicating the cost of reaching the goal from any state.
d. The optimal policy is then derived from these values, and the action that leads to the highest value in the next state is chosen for each state.

For further illustration, consider state 6 (middle of the right column) with the goal in state 9. According to the policy iteration algorithm, moving right in state 6 has a value of -1 (since it hits a wall). After policy evaluation, it becomes apparent that moving down directly toward the goal (state 9) is better. The policy is updated to move down in state 6. According to Q-iteration from state 6, the value update would immediately consider the reward of moving down towards the goal because it directly computes the maximum possible value from each action without being constrained by an initial policy.

Algorithm 2: Policy iteration

Input: T, R, γ

1. Initialize policy $\mu_0(s)$, $\forall\ s$
2. **For** k = 1: K **do**
   a. **iterate** the value function for each $s \epsilon S$ under the current fixed policy μ_k (policy evaluation step):

$$\boldsymbol{V_{i+1}^{\mu_k}(s)} = \sum_{s',r} \mathbf{Pr}(\boldsymbol{s'|s}, \boldsymbol{\mu_k(s)}) \left(\boldsymbol{R(s, \mu_k(s), s')} + \boldsymbol{\gamma V_i^{\mu_k}(s')} \right) \quad (2.28)$$

   b. Find the best action via one-step lookahead

$$\boldsymbol{\mu_{k+1}}(\boldsymbol{s}) = \underset{\boldsymbol{a}}{\mathbf{argmax}} \sum_{s',r} \mathbf{Pr}(\boldsymbol{s'|s}, \boldsymbol{a}) \left(\boldsymbol{R(s, a, s')} + \boldsymbol{\gamma V^{\mu_k}(s')} \right) \quad (2.29)$$

End for

Output: $\mu_{\mathrm{K}} = \hat{\mu}^*, V^* = V^{\mu_{\mathrm{K}}}$

For a detailed numerical example, see [82]

2.4.7. Q-Learning algorithm

There is an issue in using the Q-iteration algorithm described before. The algorithm relies on known system dynamics represented in transition function $T(s, a, s')$ and the reward acquired from this transition $R(s, a, s')$. In most nonlinear control problems, uncertainties and unknown dynamics may be presented, so a model-free algorithm is necessary. As a result, the Q-learning algorithm is proposed to deal with finite-action-space issues [83].

Q-learning is a model-free, value-based, off-policy reinforcement learning algorithm designed to identify the optimal policy within a specified environment. The term "Q" in Q-learning means “quality.” It measures the quality of the action in achieving maximum future rewards, indicating the value of the action from a given state.

Operating without a predefined model of the environment's dynamics, Q-learning approximates the optimal policy via a trial-and-error mechanism, leveraging environmental state feedback. The algorithm iteratively adjusts the Q-values, representing the expected returns from executing specific actions in certain states based on the immediate reward and the anticipated value of the following state. Through continual refinement of these Q-values informed by empirical rewards, Q-learning aims to align with an optimal policy that optimizes aggregate rewards over the duration.

In the context of value-based strategies, Q-learning focuses on enhancing the value function, which discerns the relative worth of actions within each state, thereby facilitating the selection of the most advantageous action. This iterative enhancement process, grounded in environmental reward feedback, enables the convergence toward an optimal policy that maximizes long-term rewards.

Furthermore, as an off-policy approach, Q-learning separates the policy being improved (target policy) (μ') from the behavior policy (μ) used for generating actions.

Typically, in discrete-action problems, it employs an ϵ -greedy strategy for action selection, balancing exploration and exploitation by choosing the best-known action with probability 1-ϵ time while still trying random actions with probability ϵ. This balance allows the agent to sufficiently explore the state-action space, potentially uncovering more rewarding paths.

The mathematics behind the Q-learning algorithm are described in algorithm 3:

Algorithm 3: Q-learning (Based on off-policy and Temporal difference (TD))

Input: γ, α_k where α_k is an adaptive learning rate

1. Initialize Q −function $Q_0(s,a) = 0, \forall\, s, a$
2. Measure initial state s_0
3. **For** k = 1, 2, … **do**
4. select an action a_k by using exploration strategy in the behavior policy (μ)
5. execute a_k, observe the next state s_{k+1} and reward r_{k+1}

$$\boldsymbol{Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left(r_{k+1} + \boxed{\gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k)} \right)}$$

(2.30)

The boxed part is called the TD error, representing the difference between the next target estimated return and the current estimated return. When the TD error is positive, it indicates that the action taken a_k should be chosen more frequently in subsequent decisions. Conversely, negative TD error signals, representing the inclination to select a_k, must be diminished in future situations [53]. a' is exerted from a greedy target policy μ'.

6. **End for**

2.4.8. The building blocks of the DDPG algorithm

The previously mentioned algorithms, whether model-based like Q-iteration or model-free like Q-learning, deal with finite-action-state-space problems. In addition, these methods assume the Q-functions and policies are known. In the case of the WT problem, these methods will be highly computational because our case study is a continuous-infinite-action-state space problem. In addition, the explicit definition of the Q-function and policies in continuous state-action-space is infeasible [57]. So, the previous algorithms are evolved to new model-free RL algorithms. The DDPG algorithm is recommended to achieve performance generalization and stabilization in a continuous action space without the need for mathematical modelling equations [54]. It is a simple model-free off-policy RL controller based on deep Q-Networks and actor-critic structure [55]. The term "deep" in DDPG refers to using deep neural networks as function approximators for actor and critic networks. The term "deterministic policy gradient" refers to deterministic action taken by an actor network that is updated continuously based on policies that conduct gradient ascent on a scalar objective function called the quality (value) function (Q-function) [52]. The following Subsections from (2.4.8-a) to (2.4.8-f) represent the building blocks of the DDPG algorithm.

a) Policy gradient

Policy gradient methods are a type of RL strategy that directly improves a parameterized policy $\mu(a|s,\phi^{\mu})$ through gradient ascent. These methods aim to maximize the expected return within a predefined class of policies. This is in contrast with traditional approaches that generate policies based on a value function approximation. In the policy iteration algorithm, the policy improvement step is conducted through $\underset{a}{\text{argmax}}$ of the expected return. This method suits in finite-action space problems. The problem is how to improve policy in the case of infinite-action (continuous action) space. Here, the policy gradient methods perform the policy improvement by conducting a gradient ascent on the expected return with respect to a parameterized policy.

The advantages of policy gradient methods include their ability to integrate domain-specific knowledge into the policy parameters and their typically lower parameter count needed to represent the optimal policy than those required for the corresponding value function. They are designed to reliably converge to a local optimum and are adept at managing environments with continuous states and actions, as well as scenarios with incomplete state information. However, policy gradient methods face challenges such as limited applicability in off-policy learning scenarios, slower convergence rates in discrete action spaces, and the inability to guarantee global optimality [84].

b) Actor-critic structure

Actor-critic methods utilize TD learning and incorporate a distinct memory component to represent the policy separately from the value function. This policy representation, termed the actor, is responsible for action selection, while the value function, referred to as the critic, evaluates the actions chosen by the actor. The learning process in these methods is on-policy, meaning the critic evaluates and provides feedback on the policy that the actor is currently executing. The feedback provided by the critic is in the form of a TD error, which serves as a measure of how well the actor's actions align with the expected outcomes [53].

The actor component represents the stochastic policy function $\mu_s(a|s,\phi^{\mu})$ or deterministic policy function $\mu_d(s|\phi^{\mu})$, which maps states to actions. The actor is responsible for selecting actions given the current state of the environment. The policy is parameterized by ϕ^{μ}, and the actor updates these parameters to improve policy performance based on feedback from the critic.

The critic estimates the value function, which could either be the action-value function $Q(s,a|\phi^{Q})$, or the state-value function $V(s|\phi^{V})$ at the case of stochastic policies. Regarding deterministic policies, both Q and V functions are the same. The critic evaluates the actions the actor takes by estimating how good the action is in terms of future rewards. The value function is parameterized by ϕ^{Q} or ϕ^{V}. The critic updates these parameters based on the observed rewards and the actor's actions.

The actor-critic architecture is a solution to some limitations of value-based or policy-based methods. First, policy gradient methods can suffer from high variance in

their updates, leading to potentially unstable learning processes. Value-based methods can struggle with continuous action spaces and may not inherently encourage exploration. The actor-critic method uses the critic to reduce the variance in policy updates, leading to more stable and efficient learning. Second, Actor-critic methods are particularly well-suited to environments with continuous action spaces, where the actor can output real-valued actions instead of having to select from a discrete set of actions. Thus, the computation time is reduced to pick up the actions. [53]

c) Experience Replay buffer

The experience replay buffer is a fundamental component in modern RL algorithms, especially those integrating deep learning, such as Deep Q-Networks (DQN). Its primary purpose is to store the agent's experiences at each time step, which are then used for training the model. An "experience" in this context typically comprises a tuple (s, a, r, s'), where s is the current state, a is the current action, r is the current observed reward, and s' is the next observed state after executing action a.

The motivations and functionalities of the experience replay buffer can be summarized as follows [57]:

- **Breaking Correlations**: In sequential decision-making tasks, consecutive experiences are highly correlated. Training on these correlated sequences can lead to inefficiencies and instability in learning, as the model might become biased towards the most recent experiences. The correlations are broken by storing experiences in a buffer and randomly sampling mini-batches from it, leading to more stable and robust learning.
- **Efficient Use of Experiences**: Each experience can be used multiple times in learning, making the learning process more efficient. Without experience replay, valuable experiences that could contribute significantly to learning would only influence the model once.
- **Balanced Sampling**: It allows for more uniform sampling of the state-action space over time, ensuring that rare but essential experiences are not lost but are instead used for learning alongside more common experiences.
- **Stabilizing Learning with Off-Policy Data**: Experience replay enables off-policy learning algorithms like DQN, where the learned policy differs from the policy generating the data. The buffer contains experiences generated from past policies, and the algorithm can learn from this diverse set of experiences.

In practice, the experience replay buffer is implemented as a finite-sized cache. When the cache is complete, the oldest experiences are discarded to free space for new experiences. This process ensures that the buffer reflects a mix of recent and slightly older experiences, helping the agent to remember and learn from the past. In our study, the number of episodes is 300, the number of samples per episode is 8000, and the total number of samples during the training process is 2,400,000. The cache size chosen for the experience buffer is 1,000,000 samples.

During training, instead of updating the model based on the latest experience, the algorithm randomly samples a mini-batch of experiences from the buffer. This batch then performs a gradient descent update on the model's parameters. Random sampling

helps to minimize the correlation between the updates, leading to more stable and effective learning.

d) Function Approximation

RL investigates the process by which an agent might acquire an optimum policy that maximizes their long-term reward by interacting with the environment. When there is a limited number of states and actions of moderate magnitude, the value or policy functions can be accurately shown as a table, known as the tabular setting. However, when the problem involves a vast number of states or continuous states, often in large dimensions, it becomes necessary to employ function approximation to estimate the value or policy functions involved. Due to the swift advancement of machine learning approaches in approximating functions, contemporary RL algorithms are increasingly dependent on function approximation tools to address more complicated issues. [85]

Thus, the policy (actor) and the value (critic) functions are approximated by neural network architecture to generalize the decision-making across unseen states in environments with large or continuous state spaces. The topology of the actor and critic networks is introduced later in chapter 3.

The approximation using neural networks is done by parameterizing the Q-function and policy function. $Q(s,a|\phi^Q) \approx Q^*(s,a)$ is the approximation of the true Q-function $Q^*(s,a)$ to the approximated version $Q(s,a|\phi^Q)$ parameterized by the critic neural network weights ϕ^Q. For a deterministic policy, $\mu(s|\phi^\mu) \approx \mu^*(s)$ where $\mu(s|\phi^\mu)$ is the approximated version of the true policy $\mu^*(s)$ and it is parameterized by the actor neural network weights ϕ^μ.

e) Deep Q-Network (DQN)

Deep Q-Networks (DQN) represent a significant advancement in RL, combining traditional Q-learning with deep neural networks. Introduced by DeepMind in 2015 [86], DQNs were the first to successfully apply deep learning to solve complex control tasks directly from high-dimensional sensory inputs. It is introduced with the assumption of using stochastic policy in uncertain probabilistic finite-space environments.

DQN builds on the Q-learning algorithm discussed before in Subsection (2.4.7). DQN uses a deep neural network to approximate the Q-function. The network inputs the state and outputs Q-values for all possible actions. This approach enables DQN to handle environments with high-dimensional state spaces. The training of the Q-network is conducted by minimizing consecutive loss functions $L_t(\phi^Q)$ that vary at each iteration t by performing the gradient descent algorithm with respect to the Q-network parameters ϕ^Q. The algorithm is described below.

The main contributions of this method can be summarized as follows [86]:

- **Experience Replay**: DQN stores the agent's experiences in the experience replay buffer at each step to be used in the Q-network training. The Q-learning update is performed on random mini-batches of experiences (s,a,r,s'). This helps to reduce correlations between consecutive updates and to utilize the data more efficiently.

- **Fixed Q-Targets**: For stability of training, DQN uses a separate target network to compute the target Q-values $Q'(s,a|\phi^{Q'})$, with parameters $\phi^{Q'}$. This network is a clone of the Q-network but its weights are updated less frequently.

The mathematics describing the DQN are detailed in algorithm 4:

Algorithm 4: Deep Q-network + Experience replay buffer + Target Q-network

1. Initialize the Q-network with random weights ϕ^Q
2. Initialize the experience replay buffer up to predefined number of samples
3. Initialize the target Q-network Q' with the same weights of Q $\phi^{Q'} = \phi^Q$
4. **For** episode = 1: E **do**
5. Initialize the first sequence s_1
6. **For** t = 1: T **do**
7. Select the action a_t randomly with probability ϵ, or $a_t = \max_a Q(s_t,a)|\phi^Q)$
8. Execute a_t in the environment and observe the next state s_{t+1} and observe reward r_t
9. Store the tuple (s_t, a_t, r_t, s_{t+1}) in the experience replay buffer
10. Sample a random mini-batch (s_x, a_x, r_x, s_{x+1}) of transitions from the experience replay buffer
11. set $y_x = \begin{cases} r_x & , \ for\ terminal\ s_{x+1} \\ r_x + \gamma \max_{a_{x+1}} Q'\left(s_{x+1}, a_{x+1}\middle|\phi_t^{Q'}\right), & for\ non-terminal\ s_{x+1} \end{cases}$

(2.31)

Note: a_{x+1} is the action that maximize the expected reward from the target network

12. set the loss function:

$$L_t\left(\phi_t^Q\right) = \mathbb{E}_{(s_x,a_x,r_x,s_{x+1})\sim U(D)}\left(y_x - Q\left(s_x, a_x\middle|\phi_t^Q\right)\right)^2 \quad (2.32)$$

$\mathbb{E}_{(s_x,a_x,r_x,s_{x+1})\sim U(D)}$ is the expected value of selecting random samples (tuples) from the distribution of the experience replay buffer. y_x is the target

13. The gradient descent is conducted on $L_t\left(\phi_t^Q\right)$ with respect to parameters ϕ_t^Q:

$$\nabla_{\phi_t^Q} L_t\left(\phi_t^Q\right) = \mathbb{E}_{(s_x,a_x,r_x,s_{x+1})\sim U(D)}\left(y_x - Q\left(s_x, a_x\middle|\phi_t^Q\right)\right)\nabla_{\phi_t^Q} Q\left(s_x, a_x\middle|\phi_t^Q\right)$$

(2.33)

14. Update the Q-network parameters:

$$\phi_{new}^Q = \phi_{old}^Q - \alpha\nabla_{\phi_t^Q} L_t\left(\phi_t^Q\right) \quad (2.34)$$

15. Update the target Q-network parameters every C steps: $\phi^{Q'} = \phi^Q$
16. **End for**
17. **End for**

f) Evolution to DDPG algorithm

The revolutionary DDPG algorithm [52] integrates the concepts of DQN, policy gradient, function approximation, experience replay buffer, and actor-critic methods into a unified framework designed for environments with continuous action spaces and deterministic policies.

DQN is effective in environments with high-dimensional observation spaces but is limited to discrete and low-dimensional action spaces. However, many essential tasks, particularly physical control, feature continuous (real-valued), and high-dimensional action spaces are challenging. Applying DQN directly to these continuous domains is challenging because it depends on identifying the action that maximizes the action-value function. In scenarios with continuous action values, this necessitates an iterative optimization process at each step, making straightforward application of DQN impractical. [52]

The primary innovation of DDPG lies in its successful adaptation of the advantages of DQN to continuous action domains, leveraging the actor-critic architecture for stable learning. It introduces a deterministic policy gradient approach that allows for direct optimization of policy in high-dimensional, continuous action spaces without the need for discretizing the action space or performing exhaustive search like DQN. [52]

The building blocks of the DDPG algorithm are described below:

a. **DQN (Deep Q-Network)** [86]: DQN demonstrates how deep neural networks can be used to approximate the action-value function (Q-function) effectively, handling environments with high-dimensional observation spaces. However, DQN is limited to discrete action spaces. DDPG extends this concept by using neural networks not only for the critic (value function approximation), as in the case of DQN, but also for the actor (policy function approximation).
b. **Deterministic Policy Gradient** [87]: Unlike standard policy gradient methods that optimize a stochastic policy, deterministic policy gradient methods optimize a deterministic policy. This approach simplifies the gradient computation and is more efficient for continuous action spaces since it does not require integrating over the action space [88].
c. **Actor-Critic**: The actor-critic architecture in DDPG separates the policy function (actor) and the value function (critic), allowing each to be optimized independently. The actor directly maps states to actions, facilitating learning in continuous action spaces, while the critic evaluates the selected actions, guiding the actor's improvement.
d. **Function Approximation**: DDPG uses neural networks for function approximation, enabling the algorithm to handle complex, high-dimensional environments. The actor network approximates the deterministic policy, and the critic network approximates the Q-function.
e. **Experience Replay Buffer**: as inferred from DQN, the experience replay buffer stores past transitions, which are then sampled randomly to update the actor and critic networks. This approach breaks the correlation between

consecutive learning samples, stabilizes the training, and efficiently uses experiences.

In order to prevent redundancy and repetition, the DDPG algorithm is explained in depth in Chapter 3: Controllers Design, specifically in Subsection 3.4 of the proposed F-DDPG controller design.

Chapter 3 : Controllers Design

This chapter outlines the development process of the GSPI, FMPC, the proposed CNN-C, and F-DDPG controllers. Section 3.1 represents the GSPI and details the adjustment of the PI controller parameters for each operational point. Section 3.2 describes the FMPC design and discusses the integration of fuzzy logic, the MPC prediction model, the MPC optimization process, and the control algorithm's framework. Section 3.3 details the CNN-C design and explains how supervised learning is applied in the design of the controller. Also, it presents the CNN structure, its training and optimization methodologies, and the training outcomes. Section 3.4 introduces the F-DDPG design. Also, the approach and architecture of the proposed model-free RL controller are elaborated. This includes the design of the actor and critic neural networks, the development of the reward function, a mathematical explanation of the training algorithm, the role of the fuzzy logic system, and the results achieved through training.

3.1. GSPI Controller Design

3.1.1. GSPI parameters tunning

This Subsection will discuss the methodology of tunning the GSPI Controller parameters. The main goal of the tuning is to ensure the operation of the open-loop systems, which are composed of the PI Controllers cascaded with the linearized wind-turbine models, at the same optimum cross-over frequency and phase margin (PM) [89]. The phase margin is the change permeated in the open-loop system phase without affecting the closed-loop system stability [90]. So, focusing the controller design on increasing the PM increases consequently the closed-loop system stability range. Due to the integral term of the controller, Windup causes an excursion of generator speed from the rated value and makes the pitch system unstable [91]. This problem could be mitigated by adding an anti-wind-up limiter as a part of the design to limit the pitch control action exerted by the PI Controller.

a) PI Controllers Tunning

The PI controller parameters (K_p, K_i) at average wind speed operating point of 12 m/s are tuned using MATLAB software to achieve a robust-stable-linear system with approximately the fastest rising time, least settling time, and lowest peak overshoot percentage, as shown in Table 3.1. Fixed-parameters PI controller achieves different phase margins at different cross-over frequencies for open-loop systems at different operating points. This affects the overall system relative stability [89]. So, five more PI controllers' parameters at other operating points are designed to achieve the same PM and cross-over frequency of 83.7° and 0.9 rad/s, respectively, for all open-loop systems. To schedule the gains, the tuned parameter values of all PI controllers mentioned in Table 3.2 are added into two lookup tables for K_p and K_i with breakpoints equal to average wind speeds, as illustrated in Figure 3.1. The GSPI controller parameters are

instantaneously changing online based on the current wind speed measurement. This design approach is suggested to achieve the coherence and robustness of the non-linear wind-turbine model in the face of variation of wind speed operating points, as shown in Figure 3.2, where all loop gains are closely achieving the same phase margin at the same cross-over frequency.

The sign of the gains obtained in Table 3.2 depends on how the error is defined in the system. As the error is defined as the setpoint minus the process variable, a positive error means the process variable is below the setpoint. In such a case, a negative gain might be needed to apply a corrective action in the proper direction.

Table 3.1 Performance of the tunned PI Controller at average wind speed 12 m/s

Description	Value
Rise Time (seconds)	2.07
Settling Time (seconds)	3.18
Overshoot (%)	1.32
Gain margin	14.3 dB @ 14.1 rad/s
Phase margin	83.7 deg @ 0.9 rad/s
Closed-Loop stability check	System is stable

Table 3.2 GSPI Controller best-tunned parameters at different average wind speed operating points

Average Wind Speed (m/s)	K_p	K_i
12	-0.0011653	-0.0007315
14	-0.0008615	-0.0005732
16	-0.0006988	-0.0005445
18	-0.0005842	-0.0004849
20	-0.0005079	-0.0004765
22	-0.0004498	-0.0004458

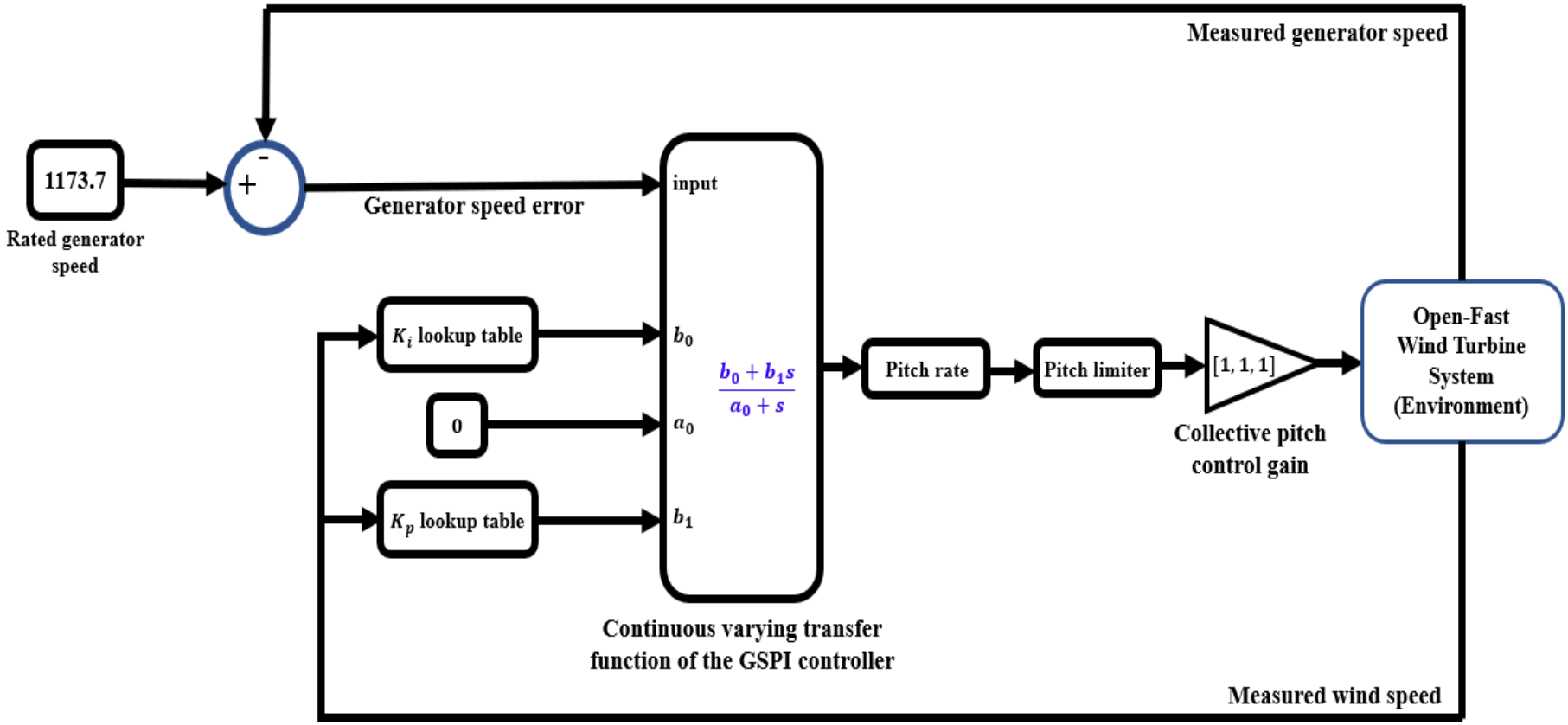


Figure 3.1 The GSPI Controller scheme

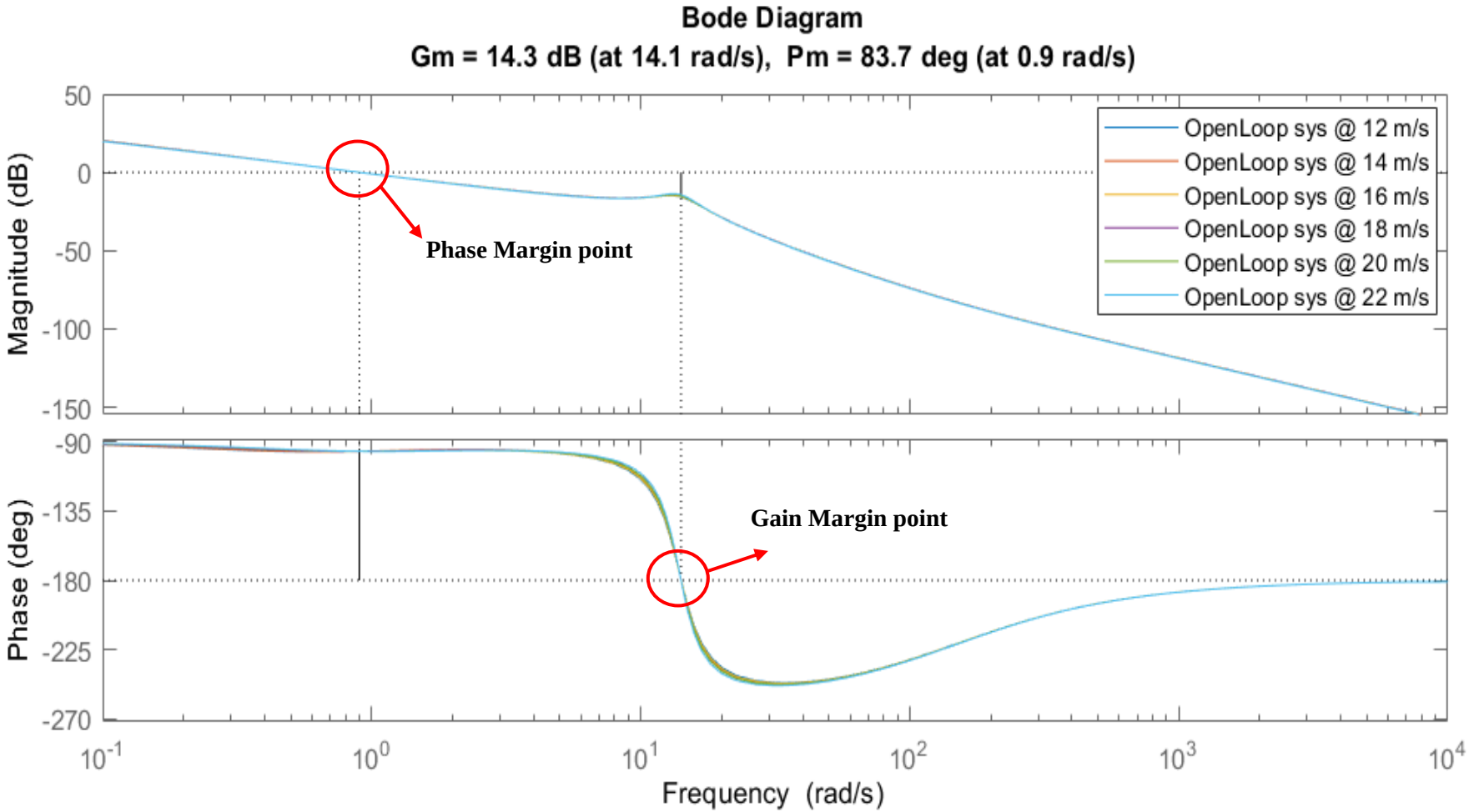


Figure 3.2 Frequency response of open loop systems with GSPI controller at different operating points

3.2. FMPC Controller Design

This study presents a model predictive controller that employs a fuzzy model to forecast upcoming outputs within a set prediction period. During each sampling moment, an online optimization task is carried out across this prediction period to determine the appropriate control actions. Only the first control action is taken based on the receding horizon approach. This optimization task is then repeated in the following sampling period. The design guidelines are inspired by [8].

Model Predictive Control (MPC) offers critical advantages:

- The predictive nature of anticipating future events makes MPC robust against various disturbances and model inaccuracies,
- MPC can handle constraints on states, input, and output, making it crucial in real-world control systems,
- MPC leverages the power of optimization techniques to find the best control action that minimizes the cost function
- MPC is a model-based controller that offers transparency and interpretability of its decision-making process.

It has been noted that many industrial controllers that predict outcomes rely on linear models. However, since industrial processes tend to be nonlinear, there is a growing inclination towards nonlinear models. Yet, as mentioned in [8], using a nonlinear model can make the optimization process more challenging. This added complexity might discourage the widespread use of nonlinear MPC in industry. Given this, we suggest using fuzzy models as a potential solution. Such models incorporate linear approaches in their outcomes but can effectively represent a nonlinear system overall. The controller is called a fuzzy model predictive controller (FMPC), and it's used as a collective pitch control action in wind turbine systems. The controller structure and the optimization technique are described in the following Subsections.

3.2.1. Collective Pitch Control Structure

The FMPC is a collective pitch angle controller for wind turbine systems to stabilize the generator speed and power operations at the rated values. The control action consists of two primary control laws, as shown in Figure 3.3. The first law relies on the steady-state average wind speed operating point of the pitch angle, which can be extracted from a lookup table based on the instantaneous wind speed. The second law is based on the FMPC for simultaneously applying collective pitch angle control to all blades.

Denote the steady state look-up table control action as u_{ss}, and the optimal collective pitch control action extracted from the FMPC as u^*_{FMPC}. Equation (3.1) represents the control action u at sampling instant (k).

$$\boldsymbol{u(k) = u_{ss}(k) + u^*_{FMPC}(k)} \tag{3.1}$$

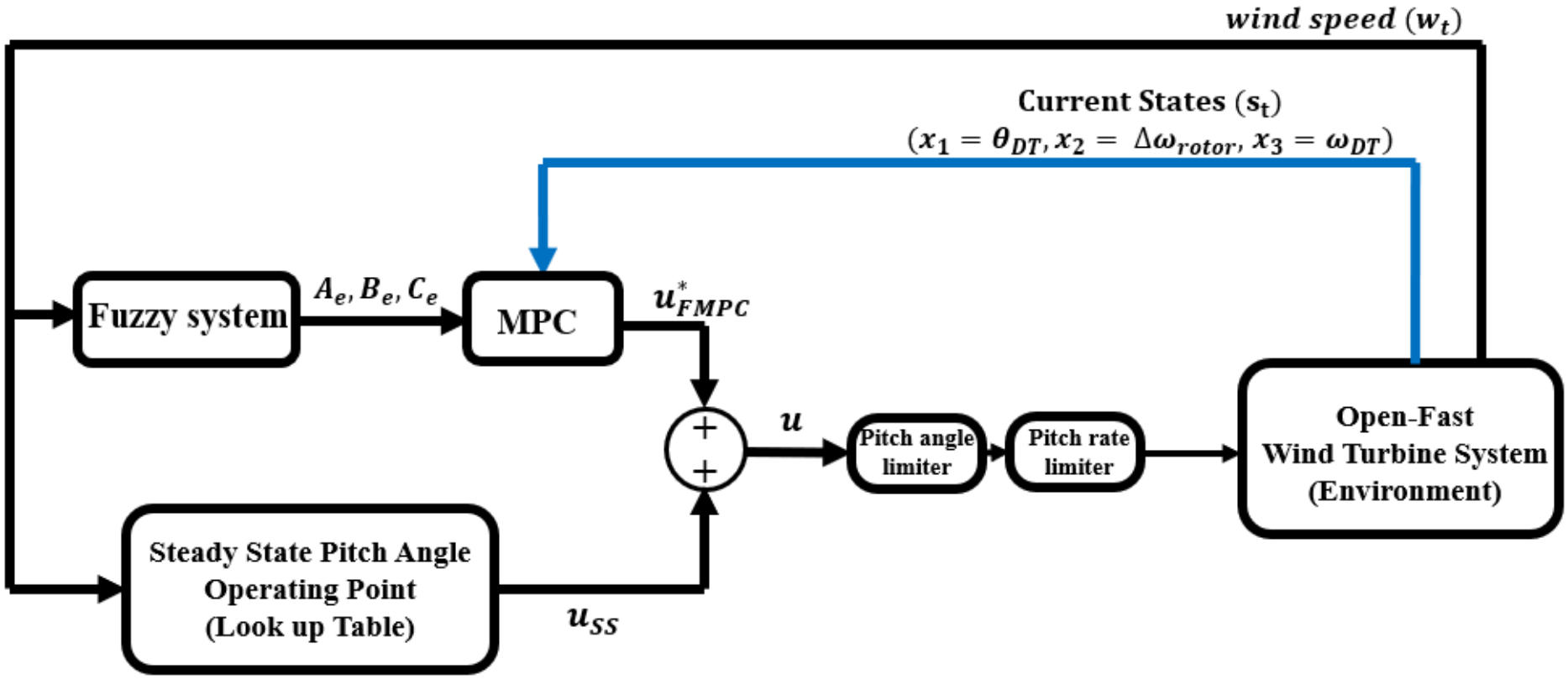


Figure 3.3 The control loop of the FMPC and the wind turbine system

3.2.2. FMPC Design

The design of the FMPC relies on three main factors: The fuzzy logic used for blending the linearized WT systems, the prediction technique to formulate the equations for the future predicted control sequence and output, and the cost function optimizer to solve the optimization problem subjected to pitch angle constraints. The fuzzy logic rule is described in Subsection (3.2.2-a). The MPC predictor is described in Subsection (3.2.2-b). The MPC optimizer is then explained in Subsection (3.2.2-c). Finally, the whole algorithm followed to calculate the control law applied to the WT is described in Subsection (3.2.2-d).

a) Fuzzy Logic manipulator

Linearized models are simplified versions of more complex systems valid around a specific operating point in a small region. However, the behavior of many systems can be non-linear and vary across different operating ranges. Multiple linearized models might be developed to manage the complexity of such systems over a wide range of operating conditions, each representing the system's behavior around a specific operating point.

The advantages of using the fuzzy logic manipulation are:

1. Fuzzy logic can be used to interpolate between linearized models, smoothly transitioning from one model to another based on the system's current state. This interpolation allows a more accurate representation of the system dynamics over a broader range of operating conditions.

2. By defining a set of fuzzy rules, the system can determine the degree to which each linearized model should be considered at a given time. Each rule corresponds to certain conditions in the system's state and prescribes a specific weighting of the respective linearized models.

3. One of the strengths of fuzzy logic is its ability to handle uncertainty and imprecision. It can combine models based on uncertain or imprecise data to produce a robust control strategy or estimate the system's state in the face of such uncertainties.

4. When controlling a non-linear system using multiple linearized models without fuzzy logic, there can be abrupt changes in control actions when switching from one model to another. Fuzzy logic helps to create smooth control surfaces, providing a gradual transition that can reduce the risk of inducing instability in the system’s response.

5. The fuzzy blending approach can make the overall system more robust to model inaccuracies because it does not rely on a single model but rather a combination that can absorb discrepancies in model predictions.

6. Instead of developing a highly complex non-linear model, multiple linear models can be developed for different operating regions. Afterwards, the

fuzzy logic can blend these to manage the overall system effectively, simplifying the design and implementation process.

The nonlinear WT system dynamics can be approximated to a linearized version using a Takagi-Sugeno fuzzy model [8]. This approach represents an accurate and straightforward form of the nonlinear model. Based on equations (2.1) and (2.2) in Subsection 2.2.4, the discretized linear models at the sampling period (T_s) and certain average wind speed (n) are represented as follows:

$$\boldsymbol{x_{dn}(k+1) = A_{dn}x_n(k) + B_{dn}u_{FMPC}(k)} \tag{3.2}$$

$$y(k) = \boldsymbol{C_{dn}x_n} \qquad n = 1, 2, \dots, 6 \tag{3.3}$$

$\boldsymbol{x_{dn}}$ is the WT state vector of discrete linear model, k is the current sampling index, y is the perturbation in generator speed, $u_{u_{FMPC}}$ is the perturbation in the collective pitch control. $\boldsymbol{A_{dn}}$, $\boldsymbol{B_{dn}}$, $\boldsymbol{C_{dn}}$ are the matrices of the discretized linear models.

According to the gap-metric criteria discussed in Subsection 2.2.4, the fuzzy logic system is used to blend the linearized models at (14, 18, and 22) m/s average wind speeds at three different rules $n = 2, 4, 6$ as shown in Figure 3.4. The fuzzy system outputs the average weighted sum of the matrices representing the linearized models based on the current wind speed measurement, as shown in (3.4).

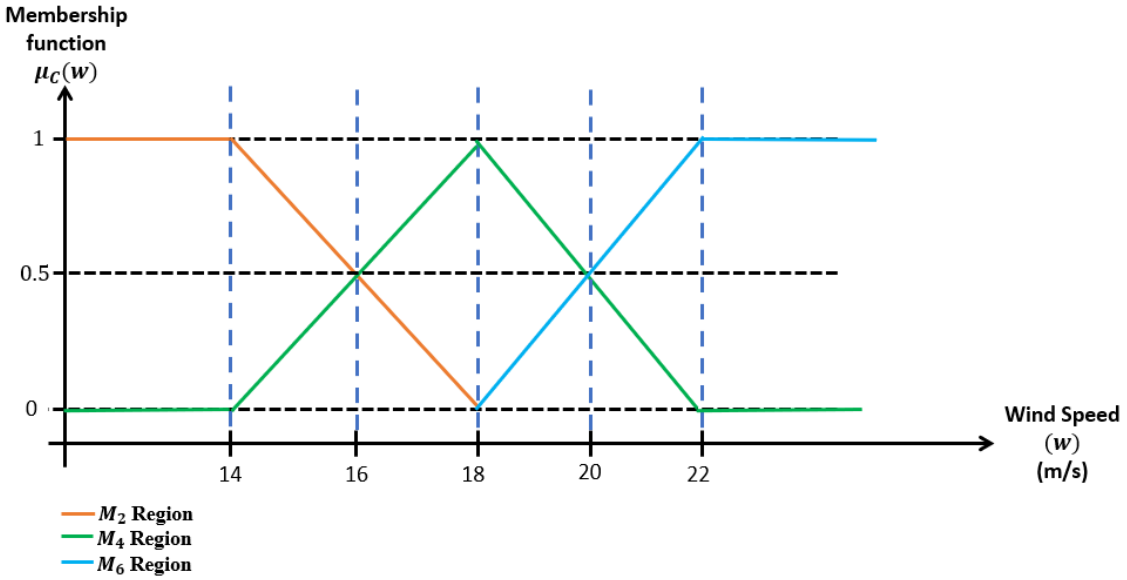


Figure 3.4 The triangular membership function representing the regions of choosing the systems

$$\boldsymbol{M_e} = \begin{cases} \boldsymbol{M_2} & , \ \boldsymbol{w < 14} \\ \boldsymbol{0.25 * \left((18 - w) * M_2 + (w - 14) * M_4\right)} & , \ \boldsymbol{14 \leq w < 18} \\ \boldsymbol{0.25 * \left((22 - w) * M_4 + (w - 18) * M_6\right)} & , \ \boldsymbol{18 \leq w < 22} \\ \boldsymbol{M_6} & , \ \boldsymbol{22 \leq w} \end{cases} \tag{3.4}$$

The notation $\boldsymbol{M_n}$ represents any of the matrices $\boldsymbol{A_{dn}}$, $\boldsymbol{B_{dn}}$ or $\boldsymbol{C_{dn}}$ at $n = 2, 4, 6$.The notation $\boldsymbol{M_e}$ is the output matrix from the fuzzy system, which represents $\boldsymbol{A_e}$, $\boldsymbol{B_e}$ or $\boldsymbol{C_e}$.

b) The MPC predictor

In MPC, the predictor is the mathematical model that forecasts future system behaviors based on current states, control actions, and disturbances. It predicts the system's evolution over a specified prediction (or control) horizon. The predictive model is crucial in MPC because it allows the controller to optimize control actions over the prediction horizon to follow a desired trajectory or setpoint while respecting imposed constraints.

The following notations are used to construct the mathematical model of the predictor. The notation $\boldsymbol{x}(k)$ is the state vector at the current sampling instant (k). The notation $\boldsymbol{x}(k+i \mid k)$ is the estimated future state vector at the i^{th} step forward based on the data at instant (k). The notation $u_{FMPC}(k)$ is the collective pitch control action output of the FMPC at instant (k). The notations for prediction and control horizons are N_p and N_c, respectively. The notation for forecasted control actions in future instances are $u_{FMPC}(k+1)$, $u_{FMPC}(k+2)$, … , $u_{FMPC}(k+N_c-1)$. The notation $y(k+i \mid k)$ is for the forecasted generator speed output at the i^{th} step forward based on the data at instant (k).

The equations (3.5) to (3.9) show the forecasted future state vectors predicted at an instant (k):

$$\boldsymbol{x(k+1 \mid k) = A_e x(k) + B_e u_{FMPC}(k)} \tag{3.5}$$

$$\boldsymbol{x(k+2 \mid k) = A_e x(k+1) + B_e u_{FMPC}(k+1)} \tag{3.6}$$

By substitution of (3.5) in (3.6), thus

$$\boldsymbol{x(k+2 \mid k) = A_e\left(A_e x(k) + B_e u_{FMPC}(k)\right) + B_e u_{FMPC}(k+1)} \tag{3.7}$$

$$\boldsymbol{x(k+2 \mid k) = A_e^2 x(k) + A_e B_e u_{FMPC}(k) + B_e u_{FMPC}(k+1)} \tag{3.8}$$

The general formula for the estimated future state vectors is

$$\boldsymbol{x}(k+i \mid k) = \boldsymbol{A_e^i}\, \boldsymbol{x}(k) + \sum_{j=1}^{i} \boldsymbol{A_e^{i-j}}\, \boldsymbol{B_e} u_{FMPC}(k+j-1) \;\; for\; i = 1,2,\dots,N_p \tag{3.9}$$

The equations (3.10) to (3.13) show the forecasted generator speed output predicted at an instant (k):

$$\boldsymbol{y(k+1 \mid k) = C_e x(k+1)} \tag{3.10}$$

$$\boldsymbol{y(k+2 \mid k) = C_e x(k+2)} \tag{3.11}$$

By substitution of (3.5) in (3.10) and (3.8) in (3.11), thus

$$y(k+1 \mid k) = \boldsymbol{C_e\left(A_e x(k) + B_e u_{FMPC}(k)\right) = C_e A_e x(k) + C_e B_e} u_{FMPC}(k) \tag{3.12}$$

$$y(k+2|\,k) = \boldsymbol{C_e A_e^2 x}(k) + \boldsymbol{C_e A_e B_e} u_{FMPC}(k) \; + \; \boldsymbol{C_e B_e} u_{FMPC}(k+1) \tag{3.13}$$

The general formula (3.14) for the estimated future output is:

$$y(k+i \mid k) = \boldsymbol{C_e A_e^i}\, \boldsymbol{x}(k) + \sum_{j=1}^{i} \boldsymbol{C_e A_e^{i-j} B_e} u_{FMPC}(k+j-1) \;\; for\; i = 1,2,\dots,N_p \tag{3.14}$$

let $N_p > N_c$, thus the control action is held constant beyond the control horizon up to the end of the prediction horizon, where $u_{FMPC}(k+N_c)$, $u_{FMPC}(k+N_c+1)$ …, $u_{FMPC}\big(k+N_p-1\big) = u_{FMPC}(k+N_c-1)$. Hence, the predicted output vector $\boldsymbol{y_F}$ can be represented as seen in (3.15):

$$\boldsymbol{y_F} = \boldsymbol{F x(k)} + \boldsymbol{\Phi u_F} \tag{3.15}$$

Where $\boldsymbol{F}$ is the state prediction matrix (state-to-output mapping matrix), $\boldsymbol{\Phi}$ is the input prediction matrix (input-to-output mapping matrix), $\boldsymbol{\Phi_1}$ is the input prediction matrix relating to the inputs from $u_{FMPC}(k)$ up to $u_{FMPC}(k+N_c-1)$, $\boldsymbol{\Phi_2}$ is the input matrix relating to the inputs from $u_{FMPC}(k+N_c)$ up to $u_{FMPC}\big(k+N_p-1\big)$, $\boldsymbol{u_F}$ is the sequence of predicted control actions vector over N_c. The matrices form of equation (3.15) are detailed in equations (3.16) to (3.18).

$$\boldsymbol{y_F} = \begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ y(k+3|k) \\ \vdots \\ y\big(k+N_p\big|k\big) \end{bmatrix}, \boldsymbol{F} = \begin{bmatrix} \boldsymbol{C_e A_e} \\ \boldsymbol{C_e A_e^2} \\ \boldsymbol{C_e A_e^3} \\ \vdots \\ \boldsymbol{C_e A_e^{N_p}} \end{bmatrix}, \boldsymbol{u_F} = \begin{bmatrix} u_{FMPC}(k) \\ u_{FMPC}(k+1) \\ u_{FMPC}(k+2) \\ \vdots \\ u_{FMPC}(k+N_c-1) \\ u_{FMPC}(k+N_c-1) \end{bmatrix}, \boldsymbol{\Phi} = [\boldsymbol{\Phi_1} \quad \boldsymbol{\Phi_2}] \tag{3.16}$$

$$\boldsymbol{\Phi_1} = \begin{bmatrix} \boldsymbol{C_e B_e} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{0} \\ \boldsymbol{C_e A_e B_e} & \boldsymbol{C_e B_e} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{C_e A_e^{N_p-1} B_e} & \boldsymbol{C_e A_e^{N_p-2} B_e} & \cdots & \boldsymbol{C_e A_e^{N_p-N_c+1} B_e} & \boldsymbol{C_e A_e^{N_p-N_c} B_e} \end{bmatrix} \tag{3.17}$$

$$\boldsymbol{\Phi_2} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \boldsymbol{0} \\ \boldsymbol{C_e B_e} & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \boldsymbol{0} \\ \boldsymbol{C_e A_e^{N_p-N_c-1} B_e} & \cdots & \boldsymbol{C_e A_e B_e} & \boldsymbol{C_e B_e} \end{bmatrix} \tag{3.18}$$

c) The MPC Optimizer

The objective of the MPC optimizer is to compute the optimal control actions over a specified prediction horizon that minimizes a particular cost function while ensuring that all model and system constraints are satisfied. This optimization is performed repeatedly at each control time step, using the latest available measurements or estimates of the system state. The MPC optimizer typically minimizes a quadratic cost function. The cost function is designed to penalize deviations of the system outputs from their desired setpoints or trajectories and to penalize excessive or aggressive control actions to prevent unnecessary wear on actuators and achieve smoother control. Hence, the equation (3.19) represents the cost function used:

$$\boldsymbol{J = \ y_F^T y_F + u_F^T \bar{R} u_F} \tag{3.19}$$

where J represents the cost function, $\boldsymbol{y_F}$ is the estimated future output vector, $\boldsymbol{u_F}$ is the estimated future input vector, $\boldsymbol{\bar{R} = zI}$ is a diagonal matrix with shape = $N_p * N_p$ and z is a tunable parameter reflecting the penalization on the control action.

The cost function is subjected to the pitch angle and pitch rate constraints. The pitch angle permissible range is typically $\left[0 - \frac{\pi}{2}\right] rad$, while the pitch rate permissible range is $[-0.139, +0.139\ ]\ rad/s$ [6]. The equations governing the constraints are represented from (3.20) to (3.25) as following:

$$\boldsymbol{u^{min} \leq u(k) \rightarrow \ u^{min} - u_{ss}(k) \leq u_{FMPC}(k)} \tag{3.20}$$

$$\boldsymbol{u^{max} \geq u(k) \rightarrow \ u^{max} - u_{ss}(k) \geq u_{FMPC}(k)} \tag{3.21}$$

$$\boldsymbol{\alpha^{min} \leq \Delta u(k) \rightarrow \ \ \alpha^{min} - \Delta u_{ss}(k) \leq \Delta u_{FMPC}(k)} \tag{3.22}$$

$$\boldsymbol{\alpha^{max} \ \geq \Delta u(k) \rightarrow \ \alpha^{max} - \Delta u_{ss}(k) \geq \Delta u_{FMPC}(k)} \tag{3.23}$$

$$\boldsymbol{\Delta u(k) = u(k) - \ u(k-1)} \tag{3.24}$$

$$\boldsymbol{\alpha^{max} = 0.139 * T_s\ , \alpha^{min} = \ -0.139 * T_s} \tag{3.25}$$

The notations for the maximum and minimum permissible pitch angles are u^{max} and u^{min}, respectively. The notations for the maximum and minimum permissible pitch angle rates are α^{max} and α^{min}, respectively.

By substituting (3.15) in (3.19), the cost function could be written as

$$\boldsymbol{J = (Fx(k) + \Phi u_F)^T (Fx(k) + \Phi u_F) + u_F^T \bar{R} u_F} \tag{3.26}$$

$$\boldsymbol{\min_{u_F} J = [Fx(k)]^T [Fx(k)] + 2u_F^T \Phi^T [Fx(k)] + u_F^T [\Phi^T \Phi + \bar{R}] u_F} \tag{3.27}$$

The objective is to find the optimal predicted control actions by minimizing the cost function with respect to the control action. The cost is subjected to the following constraints.

$$\boldsymbol{L u_F \leq M} \tag{3.28}$$

Where $\boldsymbol{L}$ and $\boldsymbol{M}$ are the constraints matrices and expressed as shown below

$$\boldsymbol{L} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{-1} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}_{4*N_p}, \boldsymbol{M} = \begin{bmatrix} \boldsymbol{-u_{ss}(k) + u^{max}} \\ \boldsymbol{u_{ss}(k) - u^{min}} \\ \boldsymbol{u_{FMPC}(k-1) - \Delta u_{ss}(k) + \alpha^{max}} \\ \boldsymbol{-u_{FMPC}(k-1) + \Delta u_{ss}(k) - \alpha^{min}} \end{bmatrix} \tag{3.29}$$

Based on the quadratic loss function and the linear constraints, the optimization problem can be solved using Hildreth's quadratic programming [92]. The Hildreth quadratic programming algorithm is a method to solve a specific type of constrained quadratic optimization problem, known as the primal problem. The algorithm is an iterative method to solve constrained quadratic optimization problems. Instead of directly solving the constrained quadratic optimization problem, Hildreth's algorithm repeatedly solves a sequence of more straightforward, unconstrained issues by transforming the primal problem into a dual problem. The unconstrained minimization problem in each iteration can be considered a kind of gradient descent step, where the algorithm moves toward the negative gradient to minimize the objective function. However, due to the constraints, this direction might be modified to ensure feasibility. By iteratively adjusting the solution and projecting it back onto the feasible set, the algorithm aims to converge to the optimal solution that minimizes the objective function and satisfies the constraints.

The optimization steps could be stated as follows:

1. Since the term $[\boldsymbol{Fx}(k)]^{\boldsymbol{T}}[\boldsymbol{Fx}(k)]$ of the loss function in (3.27) is not dependent on $\boldsymbol{u_F}$, so the quadratic loss function could be written as

$$\min_{\boldsymbol{u_F}} \boldsymbol{J} = \frac{\mathbf{1}}{\mathbf{2}} \boldsymbol{u_F^T Q u_F} + \boldsymbol{u_F^T P} \tag{3.30}$$

   Subjected to constraints $\boldsymbol{Lu_F \leq M}$

$$\text{Where } \boldsymbol{Q} = \mathbf{2} * (\boldsymbol{\Phi^T \Phi} + \boldsymbol{\bar{R}}), \boldsymbol{P} = \mathbf{2} * \boldsymbol{\Phi^T}[\boldsymbol{Fx(k)}] \tag{3.31}$$

2. The primal problem is

$$\max_{\boldsymbol{\lambda \geq 0}} \min_{\boldsymbol{u_F}} [\frac{\mathbf{1}}{\mathbf{2}} \boldsymbol{u_F^T Q u_F} + \boldsymbol{u_F^T P} + \boldsymbol{\lambda^T}(\boldsymbol{Lu_F - M})] \tag{3.32}$$

   Where $\boldsymbol{\lambda}$ is the Lagrange multipliers vector.

3. The gradient of equation (3.32) is calculated with respect to $\boldsymbol{u_F}$ being unconstrained and set to zero to obtain the following solution

$$\boldsymbol{u_F} = \boldsymbol{-Q^{-1}(P + L^T \lambda)} \tag{3.33}$$

4. Substitution of $\boldsymbol{u_F}$ (3.33) in the primal problem (3.32) yields the following dual problem.

$$\max_{\boldsymbol{\lambda \geq 0}} \left[-\frac{1}{2}\boldsymbol{\lambda^T N \lambda} - \boldsymbol{\lambda^T G} - \frac{1}{2}\boldsymbol{P^T Q^{-1} P} \right] \quad (3.34)$$

Where $\boldsymbol{N = LQ^{-1}L^T}$, $\boldsymbol{G = M + LQ^{-1}P}$ (3.35)

5. The dual problem is quadratic programming with $\boldsymbol{\lambda}$ as the decision variable and with replacing the signs in equation (3.34), this yields a minimization problem

$$\min_{\boldsymbol{\lambda \geq 0}} \left[\frac{1}{2}\boldsymbol{\lambda^T N \lambda} + \boldsymbol{\lambda^T G} + \frac{1}{2}\boldsymbol{M^T Q^{-1} M} \right] \quad (3.36)$$

6. The steps of the iterative Hildreth's algorithm to solve the dual problem are:

   a. The loss function is a quadratic function in the Lagrange multiplier λ_i^m , where i is the multiplier number, and m is the iteration number.

   b. Modify λ_i^{m+1} to minimize the objective function if $\lambda_i^{m+1} < 0 \rightarrow$ $set\ \lambda_i^{m+1} = 0$ to eliminate the inactive constraint, the equations (3.37) and (3.38) are represented as follows:

$$\boldsymbol{w_i^{m+1}} = -\frac{1}{\boldsymbol{n_{ii}}}\left[\boldsymbol{G_i} + \sum_{\boldsymbol{j=1}}^{\boldsymbol{i-1}} \boldsymbol{n_{ij}}\, \boldsymbol{\lambda_j^{m+1}} + \sum_{\boldsymbol{j=i+1}}^{\boldsymbol{n}} \boldsymbol{n_{ij}}\, \boldsymbol{\lambda_j^m}\right] \quad (3.37)$$

$$\boldsymbol{\lambda_i^{m+1}} = \mathbf{max}(\mathbf{0}, \boldsymbol{w_i^{m+1}}) \quad (3.38)$$

   Where the n_{ij} is the ij^{th} element in the matrix $\boldsymbol{N = LQ^{-1}L^T}$, and G_i is the i^{th} element in the matrix $\boldsymbol{G = M + LQ^{-1}P}$

   c. Solve for the next Lagrange component λ_{i+1}^k to get λ_{i+1}^{k+1} by repeating the steps (a) and (b).

   d. Upon convergence to the optimal solution of the Lagrange multiplier vector $\boldsymbol{\lambda^*}$, the optimal predicted control action is calculated as

$$\boldsymbol{u_F^*} = -\boldsymbol{Q^{-1}}(\boldsymbol{P} + \boldsymbol{L^T \lambda^*}) \quad (3.39)$$

   e. Choose the first component of the predicted collective pitch control action vector to be the control action of the FMPC $\boldsymbol{u_F^*} \rightarrow u_{FMPC}^*(k)$.

d) The Control algorithm

As described in this Subsection, the FMPC control algorithm relies on offline and online calculations.

The offline calculation steps are:

1. Apply the linearization on a reduced order nonlinear model of the WT at different operating points as stated in Subsection (3.2.2-a).
2. Extract the average continuous state space models from the linearization process ($\boldsymbol{A_{avg_n}}, \boldsymbol{B_{avg_n}}, \boldsymbol{C_{avg_n}}$) at the different wind speeds operating points (12, 14, 16, 18, 20, 22, 24) m/s.
3. Discretize the continuous state space models to extract the ($\boldsymbol{A_{dn}}, \boldsymbol{B_{dn}}, \boldsymbol{C_{dn}}$) matrices based on the sampling period (T_s).

4. Use the gap-metric criterion to see the correlation between the linearized models as mentioned in Table 2.3 in Subsection 2.2.4.
5. Based on the gap-metric study, select the significant minimum number of linearized models effectively representing the linearized models, which are obtained at (14, 18, 22) m/s in our case.
6. Build the fuzzy logic manipulator based on the law in equation (3.4) and triangular membership function shown in Figure 3.4.

The online calculations at each time sample are:

1. The fuzzy logic manipulator is fed with the current state of the wind speed measure and outputs the blended state space model matrices ($\boldsymbol{A_e}$, $\boldsymbol{B_e}$, $\boldsymbol{C_e}$).
2. Obtain the current measurement of the states: torsional displacement of the drivetrain (θ_{DT}), the error in rotor speed ($\Delta\omega_{rotor}$) and the torsional speed of the drivetrain (ω_{DT}).
3. The MPC optimizer, described in Subsection (3.2.2-c), solves the optimization problem in (3.27) subjected to constraints in (3.28) using Hildreth's quadratic programming algorithm to obtain the predicted optimal collective pitch control actions $\boldsymbol{u_F^*}$ based on the control horizon N_p.
4. Select the first optimal component $u_{FMPC}^*(k)$ from the control sequence $\boldsymbol{u_F^*}$ in (3.39) to form the collective pitch control part in the control law (3.1).
5. Based on the current wind speed measurement, the steady state control action u_{ss} is obtained from the look-up.
6. The control law in (3.1) is then applied to the WT system.

3.3. Proposed CNN-C Design

Artificial neural networks (ANN) start with the ability to mimic controllers' behavior for various processes as they can find complex relations between inputs and outputs. Also, it does not require previous knowledge about the procedure, which makes it outstands traditional controlling and modelling methods [7]. Standard fully connected layer neural networks (also known as multilayer perceptron), recurrent neural networks, convolutional neural networks, and Cascaded-forward fully connected layer neural networks (CNN) are all distinct types of neural network architectures designed for different tasks and data types.

Fully connected layer neural networks consist of layers of neurons where each neuron in one layer is connected to every neuron in the next layer. These networks are robust for capturing complex relationships in data but have limitations in handling high-dimensional data like images, as they do not consider the spatial hierarchy in data.

Recurrent neural networks, on the other hand, are designed to handle sequential data, such as time series or natural language. They achieve this by having connections that feed the output of a neuron back into itself, allowing them to maintain a 'memory' of previous inputs. This feedback loop provides recurrent neural networks with the capability to process sequences of data points relative to their order and context.

Convolutional neural networks exist in the field of image recognition and processing. Unlike fully connected networks, Convolutional neural networks apply convolutional filters shared across the input space, reducing the number of parameters and capturing the spatial hierarchies in the data. This allows them to efficiently handle

grid-like data and perform well on tasks such as image classification and object detection.

A CNN is a variant of the traditional feedforward neural network. Each neuron directly connects to the neurons in subsequent layers and to the output, allowing for faster learning and greater complexity as it can learn a broader range of functions. These cascading connections can lead to deeper information integration across the network.

CNN-C is a neural network controller configuring the nonlinear relations between the input features and the CPC control action output without neglecting the linear relationship between input and output [51]. In addition, it is robust and accurate against high data perturbations during network training. What makes CNN-C special is the direct link between the input and output layers and the links between hidden layers [51].

CNN-C design is based on the data acquired from FMPC operation under different mean wind speeds and high nonlinearity ordered model conditions. The CNN-C training is based on a supervised-learning approach. The main objectives of the design are:

1- To obtain a more straightforward controller that does not solve an optimization problem or calculate the blended state space model matrices ($\boldsymbol{A_e}$, $\boldsymbol{B_e}$, $\boldsymbol{C_e}$) at each time step, like the FMPC. Thus, it consumes less time during the operation compared to the FMPC.
2- To increase the confidence in applying a neural network controller in such highly non-linear and stochastic behavior WT application.
3- To generalize and enhance the performance over the FMPC by obtaining higher power output while maintaining rotor speed at its rated value.

3.3.1. CNN-C Design Methodology

CNN-C can behave like FMPC without further information about complex system dynamics and uncertainties. The goals of the design include developing an ANN controller that reduces the time required for the FMPC to solve the optimization problem. Additionally, the controller aims to enhance and generalize the performance across various stochastic behaviors associated with the WT system. These objectives are achieved through offline training, validation, and testing the CNN-C based on previously collected data. Then, the CNN-C is deployed to be tested in real-time continuous action with the WT.

The design begins with data collection, wherein the FMPC is simulated on the WT system in region three at different mean wind-speed operating points (12, 14, 16, 18, 20, 22, 24) m/s. For every operating point corresponding to a specific mean wind speed, the simulation runs for 130 seconds. At each of these points, 10,402 samples of data are gathered. When all these samples are combined across all operating points, the total count amounts to 72,814 samples.

Data are gathered to train, validate, and test the CNN-C. The data ensembled consists of seven input features and one output (label) representing the collective pitch

control action exerted by the FMPC. The features input vector ($\boldsymbol{x}^{(i)}$) at sample (i) consists of:

1. displacement of drivetrain degree of freedom (DOF) ($x_1^{(i)} = \theta_{DT}^{(i)}$)
2. error in variable speed generator DOF ($x_2^{(i)} = \Delta\omega_{rotor}^{(i)}$)
3. the velocity of drivetrain DOF ($x_3^{(i)} = \omega_{DT}^{(i)}$)
4. wind speed ($x_4^{(i)} = w^{(i)}$)
5. control action delayed by one sample ($x_5^{(i)} = u_{cpc}^{(i)}[k-1]$)
6. lookup table control action ($x_6^{(i)} = u_{ss}^{(i)}[k]$)
7. difference in lookup table control action ($x_7^{(i)} = u_{ss}^{(i)}[k] - u_{ss}^{(i)}[k-1]$)

The input features vectors and output labels of all collected samples are concatenated to form the input features matrix $\boldsymbol{X}$, and the target output vector $\boldsymbol{u_{cpc}}$. The design aims to train the CNN-C to update its parameters (weights and biases) by minimizing the sum of square error (SSE) loss function. Figure 3.5 shows a brief scheme of the data collection and the optimization of network's parameters.

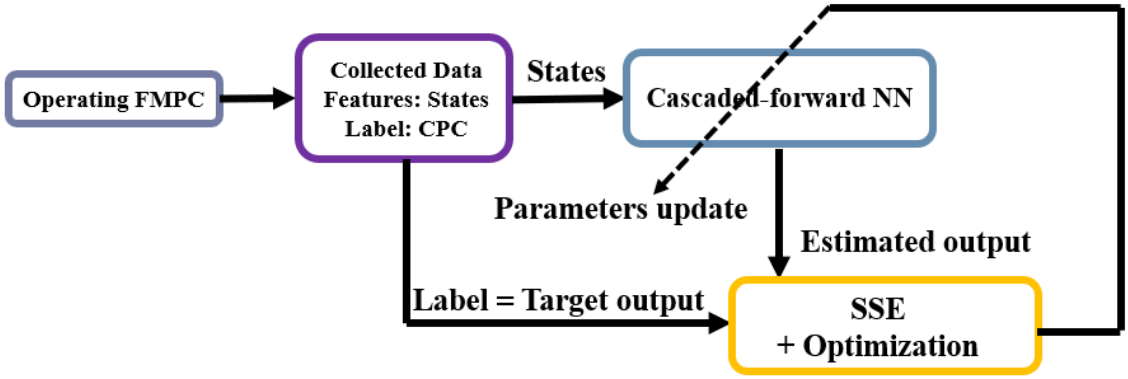


Figure 3.5 Supervised learning: CNN training and evaluation sequence

Based on the steps mentioned in the supervised learning pipeline, the block diagram is represented in Figure 3.6. shows the outlines of the supervised learning pipeline used in our task. The first step starts with data collection and preparation. The second step focuses on splitting the collected data from the operation of the FMPC to training and testing datasets. In the third step, we build an initial suggestion of the CNN model with 2 hidden layers (the number of neurons in the layers is 7 and 5). Also, the weights and biases of the neural network are initialized using random uniformly distributed values [-1,1].

In step 4, the training input features fed the suggested model to obtain the estimated model control action output. In step 5, the optimizer receives both the estimated model output and the corresponding training target output to minimize the constructed loss function with respect to the weights and biases of the model. The updates for weights and biases based on this minimization are passed to the network. Steps 4 and 5 are repeated iteratively based on a predefined number of epochs (The epoch is defined as the iteration at which all datasets are passed to the model). Parallel to steps 4 and 5, step 6 validates the model's performance on a partially selected subset of data from the training data set. In step 6, the model's performance based on a

randomly selected subset of data is tested during the training process. Step 6 is used as a stopping criterion for the model to prevent overfitting by noticing the value of the loss function after each epoch. If the value of the loss function is still decreasing, the model continues its training. If the value of the loss function is noticed to be constant or increasing for a certain predefined number of epochs, the training is terminated.

The most commonly used type of cross-validation is k-fold cross-validation [93], which works as follows:

1- Divide the training dataset into k equally-sized or approximately equal-sized subsets (folds).
2- Train your model on k-1 of these folds (the training set) and validate it on the remaining fold (validation set).
3- Repeat this process k times, using a different fold as the validation set in each iteration.
4- Calculate the performance metric (mean squared error) for each iteration.
5- Finally, average the performance metrics from all iterations to assess your model's performance.

After the training is terminated, step 7 takes place for the hyperparameters tunning process. Hyperparameter tunning is the crucial step to modify the number of layers, number of neurons, the type of activation function, regularization term if it exists, and some optimizer parameters if required. In our case, the hyperparameter tunning is limited to modifying the network's number of layers and neurons to achieve better design. Consequently, steps 4,5,6 and 7 are repeated until an accepted fined model is obtained. Moving to step 8, the best-designed model is tested using a new unseen testing dataset. The testing input features are passed to the model to get the estimated model output. In step 9, the model evaluation process takes place, where the model estimated output is compared with the testing data target output to get the evaluated loss function value.

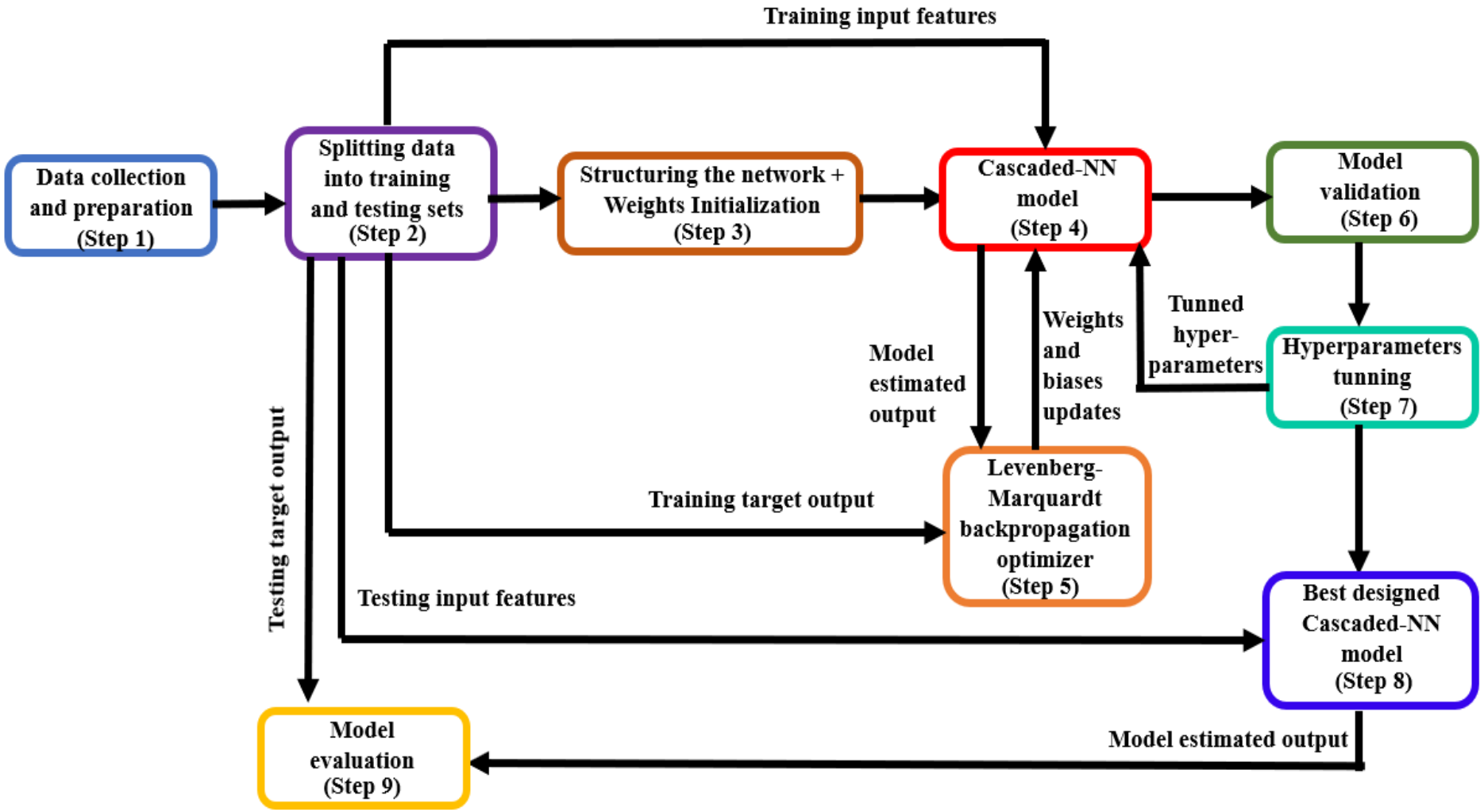


Figure 3.6 Supervised learning: CNN training and evaluation sequence

3.3.2. CNN Topology

The choice of the number of hidden layers and neurons in each layer is a hyperparameter to be tuned. The selection is based initially on the recommendation in [94]. The selection criterion starts with one or two hidden layers. Then, it increases the number of layers to increase the complexity of feature extraction from the data. Initially, choose the number of neurons in the first layer with the same number of input features. Based on the previous recommendation, we start configuring the layers and neurons and then validate the model iteratively. A choice of typically four hidden layers is found to be sufficient, and the acquired results are satisfactory.

The network structure consists of one feature input layer, four hidden layers, and one output layer, as shown in Figure 3.7. The four hidden layers consist of (20, 15, 10, and 5) neurons. The activation (transfer functions) imported following each hidden layer is tanh function.

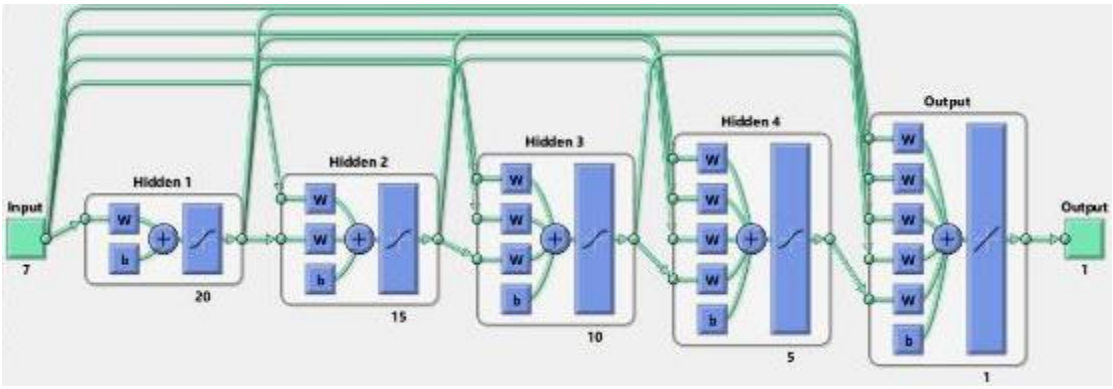


Figure 3.7 CNN Topology

In forward propagation, the input features travel through the network, moving from the input layer through all the hidden layers in a left-to-right sequence until they reach the output layer. The pitch control action is determined at the output layer based on the predicted label. The equations governing the forward propagation process are introduced from (3.40) to (3.49). The cascading property appears in these equations as the input to hidden layers contains all previous hidden layers' outputs up to the input feature matrix $\boldsymbol{X}$.

$$\boldsymbol{Z}^{[1]} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{X}}^{[1]} + \boldsymbol{b}^{[1]} \tag{3.40}$$

$$\boldsymbol{A}^{[1]} = \mathbf{tanh}\,(\boldsymbol{Z}^{[1]}) \tag{3.41}$$

$$\boldsymbol{Z}^{[2]} = \boldsymbol{A}^{[1]}\boldsymbol{W}_{\boldsymbol{A1}}^{[2]} + \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{X}}^{[2]} + \boldsymbol{b}^{[2]} \tag{3.42}$$

$$\boldsymbol{A}^{[2]} = \tanh\,(\boldsymbol{Z}^{[2]}) \tag{3.43}$$

$$\boldsymbol{Z}^{[3]} = \boldsymbol{A}^{[2]}\boldsymbol{W}_{\boldsymbol{A2}}^{[3]} + \boldsymbol{A}^{[1]}\boldsymbol{W}_{\boldsymbol{A1}}^{[3]} + \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{X}}^{[3]} + \boldsymbol{b}^{[3]} \tag{3.44}$$

$$\boldsymbol{A}^{[3]} = \tanh\,(\boldsymbol{Z}^{[3]}) \tag{3.45}$$

$$\boldsymbol{Z}^{[4]} = \boldsymbol{A}^{[3]}\boldsymbol{W}_{\boldsymbol{A3}}^{[4]} + \boldsymbol{A}^{[2]}\boldsymbol{W}_{\boldsymbol{A2}}^{[4]} + \boldsymbol{A}^{[1]}\boldsymbol{W}_{\boldsymbol{A1}}^{[4]} + \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{X}}^{[4]} + \boldsymbol{b}^{[4]} \tag{3.46}$$

$$\boldsymbol{A}^{[4]} = \tanh(\boldsymbol{Z}^{[4]}) \tag{3.47}$$

$$\boldsymbol{Z}^{[5]} = \boldsymbol{A}^{[4]}\boldsymbol{W}_{\boldsymbol{A4}}^{[5]} + \boldsymbol{A}^{[3]}\boldsymbol{W}_{\boldsymbol{A3}}^{[5]} + \boldsymbol{A}^{[2]}\boldsymbol{W}_{\boldsymbol{A2}}^{[5]} + \boldsymbol{A}^{[1]}\boldsymbol{W}_{\boldsymbol{A1}}^{[5]} + \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{X}}^{[5]} + \boldsymbol{b}^{[5]} \tag{3.48}$$

$$\widehat{\boldsymbol{u}}_{\boldsymbol{cpc}} = \mathbf{linear}(\boldsymbol{Z}^{[5]}) \tag{3.49}$$

where $\boldsymbol{Z}^{[l]}$ is the linear output matrix of the hidden layer $[l]$, $\boldsymbol{W}_{\boldsymbol{X}}^{[l]}$ is the weight matrix connecting input layer matrix $\boldsymbol{X}$ to the neurons of layer $[l]$, $\boldsymbol{b}^{[l]}$ is the bias vector at layer $[l]$. $\boldsymbol{A}^{[l]}$ is the output matrix of the tanh activation function at layer $[l]$, $\boldsymbol{W}_{\boldsymbol{A_{l-m}}}^{[l]}$ is the weight matrix connecting neurons of layer $[l]$ with the neurons of layer $[l-m]$, $\widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}$ is the estimated collective pitch output vector at all samples.

Table 3.3 shows different neural network parameter matrices' shapes (sizes). Features input matrix ($\boldsymbol{X}$) shape is (N_S, N_X) where N_S and N_X are the number of samples and number of features, respectively, i.e. $(72814,7)$. The shape of $\boldsymbol{W}_{\boldsymbol{A_{l-m}}}^{[l]}$ is (N_{l-m}, N_l) where N_{l-m} and N_l are the number of neurons in layers $l-m$ and l, respectively. The shape of $\boldsymbol{W}_{\boldsymbol{X}}^{[l]}$ is (N_X, N_l). The shape of $\boldsymbol{b}^{[l]}$ is (1, N_l). The shape of $\boldsymbol{Z}^{[l]}$ is (N_S, N_l).The shape of $\boldsymbol{A}^{[l]}$ is of the same shape as $\boldsymbol{Z}^{[l]}$. Based on the shapes of the weight matrices, the total number of parameters (weights and biases) in the CNN could be calculated by summing the multiplication of the number of rows and columns in each weight matrix as follows:

$$Total\ Number\ of\ parameters = (7*20)_{\boldsymbol{W}_{\boldsymbol{X}}^{[1]}} + (20)_{\boldsymbol{b}^{[1]}} + (7*15)_{\boldsymbol{W}_{\boldsymbol{X}}^{[2]}} + (20*15)_{\boldsymbol{W}_{\boldsymbol{A1}}^{[2]}} + (15)_{\boldsymbol{b}^{[2]}} \ldots + (1*1)_{\boldsymbol{b}^{[5]}} = 1333 \tag{3.50}$$

The following shows the construction of network predefined matrices.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \boldsymbol{x}_2^{(1)} & \ldots & \boldsymbol{x}_{N_X}^{(1)} \\ \boldsymbol{x}_1^{(2)} & \boldsymbol{x}_2^{(2)} & \ldots & \boldsymbol{x}_{N_X}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_1^{(N_S)} & \boldsymbol{x}_2^{(N_S)} & \ldots & \boldsymbol{x}_{N_X}^{(N_S)} \end{bmatrix}, \boldsymbol{W}_{\boldsymbol{X}}^{[l]} = \begin{bmatrix} \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{1},\boldsymbol{1}) & \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{1},\boldsymbol{2}) & \ldots & \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{1},\boldsymbol{N_l}) \\ \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{2},\boldsymbol{1}) & \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{2},\boldsymbol{2}) & \ldots & \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{2},\boldsymbol{N_l}) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{N_X},\boldsymbol{1}) & \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{N_X},\boldsymbol{2}) & \ldots & \boldsymbol{w}_{\boldsymbol{X}}^{[l]}(\boldsymbol{N_X},\boldsymbol{N_l}) \end{bmatrix} \tag{3.51}$$

$$\boldsymbol{W}_{\boldsymbol{A_{l-m}}}^{[l]} = \begin{bmatrix} \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{1},\boldsymbol{1}) & \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{1},\boldsymbol{2}) & \ldots & \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{1},\boldsymbol{N_l}) \\ \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{2},\boldsymbol{1}) & \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{2},\boldsymbol{2}) & \ldots & \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{2},\boldsymbol{N_l}) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{N_{l-m}},\boldsymbol{1}) & \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{N_{l-m}},\boldsymbol{2}) & \ldots & \boldsymbol{w}_{\boldsymbol{A_{l-m}}}^{[l]}(\boldsymbol{N_{l-m}},\boldsymbol{N_l}) \end{bmatrix}, \boldsymbol{b}^{[l]} = \begin{bmatrix} \boldsymbol{b}_{\boldsymbol{1}}^{[l]} \\ \boldsymbol{b}_{\boldsymbol{2}}^{[l]} \\ \vdots \\ \boldsymbol{b}_{\boldsymbol{N_l}}^{[l]} \end{bmatrix}^{\boldsymbol{T}} \tag{3.52}$$

$$\boldsymbol{Z}^{[l]} = \begin{bmatrix} \boldsymbol{z}_1^{[l](1)} & \boldsymbol{z}_2^{[l](1)} & \ldots & \boldsymbol{z}_{N_l}^{[l](1)} \\ \boldsymbol{z}_1^{[l](2)} & \boldsymbol{z}_2^{[l](2)} & \ldots & \boldsymbol{z}_{N_l}^{[l](2)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{z}_1^{[l](N_s)} & \boldsymbol{z}_2^{[l](N_s)} & \ldots & \boldsymbol{z}_{N_l}^{[l](N_s)} \end{bmatrix}, \boldsymbol{A}^{[l]} = \begin{bmatrix} \boldsymbol{a}_1^{[l](1)} & \boldsymbol{a}_2^{[l](1)} & \ldots & \boldsymbol{a}_{N_l}^{[l](1)} \\ \boldsymbol{a}_1^{[l](2)} & \boldsymbol{a}_2^{[l](2)} & \ldots & \boldsymbol{a}_{N_l}^{[l](2)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{a}_1^{[l](N_s)} & \boldsymbol{a}_2^{[l](N_s)} & \ldots & \boldsymbol{a}_{N_l}^{[l](N_s)} \end{bmatrix} \tag{3.53}$$

Biasing vectors adds an additional degree of freedom for the network model to fit the data upward or downward in the linear output equation [95]. MATLAB has a feature that broadcasts the values of bias vectors over all the operations for all samples, making it computationally efficient. Various activation functions were evaluated in the hidden layers. The tanh function yielded the most favorable outcomes, as indicated by the lowest cost function value. The tanh function's effectiveness stems from its advantages over the sigmoid function and various versions of Rectified Linear Unit (ReLU) activation functions. Specifically, the tanh function offers more flexible gradient changes, is zero-centered, and has an output range between -1 and 1, including negative values. The vanishing gradient problem during optimization is avoided due to the small values of the hidden layers activated output that lies almost in the linear region of the tanh function. [96]

Table 3.3 Dimensions and shapes of the CNN matrices

Layer type	**Inputs**	**Network Weights** $W^{[l]}_{multiplied\ matrix}$	**layers biasing vector** $b^{[l]}$	**Hidden layers linear outputs** $Z^{[l]}$	**layers activated outputs** $A^{[l]}$
First Hidden Layer $l = 1$ (20 neurons)	$\boldsymbol{X}$	$\boldsymbol{W_X^{[1]}} = (7, 20)$	$\boldsymbol{b^{[1]}} = (1,20)$	$\boldsymbol{Z^{[1]}} = (72814,20)$	$\boldsymbol{A^{[1]}} = (72814, 20)$
Second Hidden Layer $l = 2$ (15 neurons)	$\boldsymbol{X}$ $\boldsymbol{A^{[1]}}$	$\boldsymbol{W_X^{[2]}} = (7,15)$ $\boldsymbol{W_{A1}^{[2]}} = (20,15)$	$\boldsymbol{b^{[2]}} = (1,15)$	$\boldsymbol{Z^{[2]}} = (72814,15)$	$\boldsymbol{A^{[2]}} = (72814,15)$
Third Hidden Layer $l = 3$ (10 neurons)	$\boldsymbol{X}$ $\boldsymbol{A^{[1]}}$ $\boldsymbol{A^{[2]}}$	$\boldsymbol{W_X^{[3]}} = (7,10)$ $\boldsymbol{W_{A1}^{[3]}} = (20,10)$ $\boldsymbol{W_{A2}^{[3]}} = (15,10)$	$\boldsymbol{b^{[3]}} = (1,10)$	$\boldsymbol{Z^{[3]}} = (72814,10)$	$\boldsymbol{A^{[3]}} = (72814,10)$
Fourth Hidden Layer $l = 4$ (5 neurons)	$\boldsymbol{X}$ $\boldsymbol{A^{[1]}}$ $\boldsymbol{A^{[2]}}$ $\boldsymbol{A^{[3]}}$	$\boldsymbol{W_X^{[4]}} = (7,5)$ $\boldsymbol{W_{A1}^{[4]}} = (20,5)$ $\boldsymbol{W_{A2}^{[4]}} = (15,5)$ $\boldsymbol{W_{A3}^{[4]}} = (10,5)$	$\boldsymbol{b^{[4]}} = (1,5)$	$\boldsymbol{Z^{[4]}} = (72814,5)$	$\boldsymbol{A^{[4]}} = (72814,5)$
Output Layer $l = 5$ (1 neuron)	$\boldsymbol{X}$ $\boldsymbol{A^{[1]}}$ $\boldsymbol{A^{[2]}}$ $\boldsymbol{A^{[3]}}$ $\boldsymbol{A^{[4]}}$	$\boldsymbol{W_X^{[5]}} = (7,1)$ $\boldsymbol{W_{A1}^{[5]}} = (20,1)$ $\boldsymbol{W_{A2}^{[5]}} = (15,1)$ $\boldsymbol{W_{A3}^{[5]}} = (10,1)$ $\boldsymbol{W_{A4}^{[5]}} = (5,1)$	$\boldsymbol{b^{[5]}} = (1,1)$	$\boldsymbol{Z^{[5]}} = (72814,1).$	$\boldsymbol{\widehat{U}_{cpc}} = (72814,1)$

3.3.3. Training and optimization

The CNN training is based on supervised learning. The sequence of the supervised training is described in the following steps. First, the data are collected, preprocessed by shuffling, and split into training and testing datasets. The model is fed with a training dataset, which consists of the input features $\boldsymbol{X}$ and the target (label) (control output) u_{cpc}.The input features $\boldsymbol{X}$ are forward propagated to the CNN to output the estimated control output $\hat{u}_{cpc}$. Third, the SSE loss function L_{CNN} is calculated based on the target and estimated control outputs. Fourth, the optimization of the loss function is done by minimizing L_{CNN} using the Levenberg-Marquardt back-propagation (LMB) algorithm [97]. Fifth, the network parameters are updated based on the gradient. Sixth, the model performance validity is tested using the testing dataset.

a) Data preprocessing

The steps followed to prepare the data for training are explained in the following steps. First, the data are collected from running FMPC at different wind speed operating points. At each of these points, 10,402 samples of data are gathered. Second, the matrices containing the samples are concatenated sequentially to form a data matrix of 72814 examples (samples). The data matrix contains both X and u_{cpc}. Third, the data matrix is shuffled randomly to increase the diversity between the samples' correlations, improve the learning capabilities, and generalize the model performance. Table 3.4 shows the head part of the shuffled data structure. Fourth, the data are split into training and testing datasets with 80 % and 20 %, respectively. Fifth, the data matrix is split into $\boldsymbol{X}$ matrix and $\boldsymbol{u_{cpc}}$ vector.

Table 3.4 Head part of the gathered data structure.

index	Input features $\boldsymbol{X}$							Target Control Action
(i)	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$	$x_5^{(i)}$	$x_6^{(i)}$	$x_7^{(i)}$	$u_{cpc}^{(i)}$
1	-0.0001	-0.0054	0.0058	22.1314	-0.0375	0.3473	0.013	-0.0488
2	0.0006	-0.0063	0.0037	23.5126	0.0319	0.374	-0.0007	0.0343
3	-0.0001	0.0001	-0.0037	12.9334	0.0289	0.1055	0.0015	0.0256
4	-0.0002	0.0008	-0.0025	13.7858	-0.0011	0.141	0.0076	-0.0104
5	0	-0.0005	-0.0014	18.4694	-0.0162	0.2695	-0.0032	-0.0147

Note: the $u_{cpc}^{(i)}$ is the collective pitch control action exerted directly from the FMPC before passing through pitch angle and rate limiters.

b) LMB Optimizer

The loss function is minimized by adjusting the network weights and biases. In the domain of neural network optimization, the choice of an algorithm is pivotal to the model's convergence and performance. Table 3.5 shows a comparison between different optimization algorithms. The main corner points of the comparison are the methodology, advantages, disadvantages, and learning rate adjustment. The mentioned algorithms in Table 3.5 are the LMB algorithm, batch gradient descent algorithm, stochastic gradient descent, Minibatch gradient descent, adaptive gradient optimizer (ADAGRAD)[98], and adaptive moment estimation (ADAM) [99].

The batch gradient descent algorithm emerges as a stable yet computationally intensive choice for expansive datasets. The stochastic gradient descent algorithm updates parameters incrementally for each training example, enabling faster but less consistent and noisy convergence. Minibatch gradient descent balances the two previously mentioned algorithms, offering improved computational efficiency while dampening noise in the parameter updates [100].

Advanced adaptations like ADAGRAD and ADAM bring adaptive learning rate mechanisms to the table, with ADAM further incorporating momentum by tracking the exponential moving average of gradients. These methods expedite convergence across complex error surfaces typical of large-scale neural networks [98], [99].

Despite these advancements, LMB's nuanced approach to parameter updates, which leverages the curvature of the error landscape, often outperforms these alternatives in scenarios with a few hundred neural network parameters [97]. This proficiency arises from its calculated step sizes in parameter space, navigating more directly toward the objective minimum, even if its inability to scale to larger networks is due to increased computing demand.

The LMB algorithm is particularly distinguished in training networks with a limited number of parameters, typically in the hundreds or one or two thousand. It employs an adept blend of gradient descent and Gauss-Newton methods, allowing it to converge rapidly using both first-order and approximate second-order derivative information [97]. It is particularly known for its application in curve fitting. When applied to the training of neural networks, it serves as a rapid training method [101]. The algorithm is named after Kenneth Levenberg and Donald Marquardt, who proposed it in the context of non-linear least squares minimization.

The idea behind the optimization process is to train the CNN to update its parameters (weights and biases) by minimizing the sum of square error (SSE) loss function L_{CNN} described in equation (3.56).

Table 3.5 Comparison between different types of most popular used optimization techniques.

Algorithm / Feature	Levenberg-Marquardt (LMB)	Batch Gradient Descent	Stochastic Gradient Descent	Minibatch Gradient Descent	ADAGRAD	ADAM
Methodology	A second-order method that combines gradient descent and Gauss-Newton. The update is derived from a linear approximation of the error surface.	Updates weights using the gradient of the entire dataset.	Updates weights using the gradient of a single randomly picked training example.	A compromise between batch and stochastic methods. Updates weights using the gradient of a random subset of the training data.	Uses a component-wise adaptive learning rate derived from squared gradients.	Combines momentum and adaptive learning rates, using estimates of first and second moments of the gradients.
Advantages	Rapid convergence when close to solution. Suitable for problems where the Hessian can be approximated.	Stable convergence due to consistent gradient direction. Efficient matrix operations possible.	Can escape shallow local minima due to noisy updates. Often faster convergence because of frequent updates.	Balances speed of SGD and stability of batch gradient descent. Can take advantage of matrix optimizations.	Adaptively scales the learning rate, which can be beneficial for sparse data or features.	Robust and performs well in various settings. It is often used as a default optimizer in deep learning.

Disadvantages	Computationally and memory intensive due to matrix operations and inversions. It is not suitable for huge datasets or high-dimensional problems.	It can be very slow for large datasets. Stuck at shallow local minima.	Noisy updates can lead to divergence or oscillation.	It requires the selection of a good minibatch size.	The learning rate can decrease to minimal values, potentially stopping learning prematurely.	It still has hyperparameters that might need tuning (though often default values work well).
Learning Rate Adjustment	It is adjusted via the damping factor, which behaves similarly to a learning rate.	Typically fixed, but can be adjusted with methods like learning rate annealing.	It often starts high and gets annealed over time.	It is the same as SGD, often annealed over time.	It is adjusted adaptively for each parameter based on past gradients.	It is adjusted adaptively for each parameter.

i. Levenberg-Marquardt modification

$$\widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}^{(\boldsymbol{i})} = \boldsymbol{f}\left(\boldsymbol{x}_{\boldsymbol{1}}^{(\boldsymbol{i})}, \boldsymbol{x}_{\boldsymbol{2}}^{(\boldsymbol{i})}, \dots, \boldsymbol{x}_{\boldsymbol{P}}^{(\boldsymbol{i})} | \boldsymbol{w}_{\boldsymbol{1}}, \boldsymbol{w}_{\boldsymbol{2}}, \dots, \boldsymbol{w}_{\boldsymbol{M}}\right) = \boldsymbol{f}(\boldsymbol{x}^{(\boldsymbol{i})} | \boldsymbol{w}) \tag{3.54}$$

$$\boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{B}} \left(\boldsymbol{u}_{\boldsymbol{cpc}}^{(\boldsymbol{i})} - \widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}^{(\boldsymbol{i})}\left(\boldsymbol{x}^{(\boldsymbol{i})} | \boldsymbol{w}\right)\right)^{\boldsymbol{2}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{B}} \boldsymbol{e}_{\boldsymbol{i}}(\boldsymbol{w})^{\boldsymbol{2}} \tag{3.55}$$

$$\boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w}) = \left|\boldsymbol{u}_{\boldsymbol{cpc}} - \widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}(\boldsymbol{X}|\boldsymbol{w})\right|^{\boldsymbol{T}} \left|\boldsymbol{u}_{\boldsymbol{cpc}} - \widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}(\boldsymbol{X}|\boldsymbol{w})\right| = \boldsymbol{e}(\boldsymbol{w})^{\boldsymbol{T}} \boldsymbol{e}(\boldsymbol{w}) \tag{3.56}$$

Assume for notation simplicity that ($\boldsymbol{w}$) is a flattened vector that contains all network parameters (weights and biases). Where i is the sample (example) number, $P = 7$ is the total number of features, w_j is the j number weight, M is the total number of parameters in the CNN, B is the mini-batch size, $u_{cpc}^{(i)}$ is the targeted FMPC collective pitch control action at sample i, $\hat{u}_{cpc}^{(i)}(\boldsymbol{x}^{(\boldsymbol{i})}|\boldsymbol{w})$ is the estimated collective pitch control output exerted from the CNN by feedforwarding the network with input feature states vector ($\boldsymbol{x}^{(\boldsymbol{i})}$) at sample i based on ($\boldsymbol{w}$).

The residual error $e_i(\boldsymbol{w})$ at sample i is based on network parameters ($\boldsymbol{w}$), and represented as following

$$\boldsymbol{e}_{\boldsymbol{i}}(\boldsymbol{w}) = \boldsymbol{u}_{\boldsymbol{cpc}}^{(\boldsymbol{i})} - \widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}^{(\boldsymbol{i})}\left(\boldsymbol{x}^{(\boldsymbol{i})}|\boldsymbol{w}\right) \tag{3.57}$$

The vectorized version of the loss function is described in equation (3.56), where $\boldsymbol{u}_{\boldsymbol{cpc}}$ represents the target collective pitch control vector and $\widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}(\boldsymbol{X}|\boldsymbol{w})$ represents the estimated collective pitch control output vector from the neural network by forwarding the input feature matrix $\boldsymbol{X}$. $\boldsymbol{e}$ is the residual error vector

$$\boldsymbol{e}(\boldsymbol{w}) = \boldsymbol{u}_{\boldsymbol{cpc}} - \widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}(\boldsymbol{X}|\boldsymbol{w}) = [\boldsymbol{e}_{\boldsymbol{1}}\ \boldsymbol{e}_{\boldsymbol{2}} \dots \boldsymbol{e}_{\boldsymbol{B}}]^{\boldsymbol{T}} \tag{3.58}$$

According to [101], the LMB is an iterative algorithm approximating Newton's method. The objective of training the neural network at each iteration is to find the weights update ($\boldsymbol{\Delta w}$) to obtain the optimal weights at this iteration finally ($\boldsymbol{w}^* = \boldsymbol{w} + \boldsymbol{\Delta w}$) to minimize the loss function at the next iteration $L_{CNN}(\boldsymbol{w} + \boldsymbol{\Delta w})$. The loss function is approximated using second-order Taylor expansion to clarify the algorithm.

$$L_{CNN}(\boldsymbol{w} + \boldsymbol{\Delta w}) \approx L_{CNN}(\boldsymbol{w}) + [\nabla \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w})]^T \boldsymbol{\Delta w} + \frac{1}{2} \boldsymbol{\Delta w}^{\boldsymbol{T}} \nabla^{\boldsymbol{2}} \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w}) \boldsymbol{\Delta w} \tag{3.59}$$

Where $\nabla^{\boldsymbol{2}} \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w})$ is the Hessian matrix of the loss function and $\nabla \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w})$ is the gradient matrix. If $\nabla^{\boldsymbol{2}} \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w})$ is a positive-definite matrix; thus, the 2nd-order approximation is a concave-up function of $\boldsymbol{\Delta w}$, and the function $\boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w} + \boldsymbol{\Delta w})$ minimum point can be found by setting the derivative with respect to $\boldsymbol{\Delta w}$ to zero.

$$0 = \frac{\partial}{\partial \boldsymbol{\Delta w}} [L_{CNN}(\boldsymbol{w}) + [\nabla \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w})]^T \boldsymbol{\Delta w} + \frac{1}{2} \boldsymbol{\Delta w}^{\boldsymbol{T}} \nabla^{\boldsymbol{2}} \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w}) \boldsymbol{\Delta w}] \tag{3.60}$$

According to Newton's method, this would result in the following:

$$\boldsymbol{\Delta w} = -[\nabla^{\boldsymbol{2}} \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w})]^{-1} \nabla \boldsymbol{L}_{\boldsymbol{CNN}}(\boldsymbol{w}) \tag{3.61}$$

Since the loss function is the sum of square errors, it would show that:

$$\nabla \boldsymbol{L_{CNN}}(\boldsymbol{w}) = \mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\boldsymbol{e}(\boldsymbol{w}) \tag{3.62}$$

$$\nabla^{\mathbf{2}} \boldsymbol{L_{CNN}}(\boldsymbol{w}) = \mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\mathbf{J}(\boldsymbol{w}) + \boldsymbol{S}(\boldsymbol{w}) \tag{3.63}$$

where $\mathbf{J}(\boldsymbol{w})$ is the Jacobian matrix, the rows of the Jacobian matrix are the first partial derivative of the residual error $e_i(\boldsymbol{w})$ at each sample of the mini-batch size. The columns of $\mathbf{J}(\boldsymbol{w})$ represent the partial derivative of the residual error with respect to a certain parameter of the network (w_j). The Jacobian matrix is represented as the following:

$$\mathbf{J}(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial \boldsymbol{e_1}(\boldsymbol{w})}{\partial \boldsymbol{w_1}} & \frac{\partial \boldsymbol{e_1}(\boldsymbol{w})}{\partial \boldsymbol{w_2}} & \cdots & \frac{\partial \boldsymbol{e_1}(\boldsymbol{w})}{\partial \boldsymbol{w_M}} \\ \frac{\partial \boldsymbol{e_2}(\boldsymbol{w})}{\partial \boldsymbol{w_1}} & \frac{\partial \boldsymbol{e_2}(\boldsymbol{w})}{\partial \boldsymbol{w_2}} & \cdots & \frac{\partial \boldsymbol{e_2}(\boldsymbol{w})}{\partial \boldsymbol{w_M}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{e_B}(\boldsymbol{w})}{\partial \boldsymbol{w_1}} & \frac{\partial \boldsymbol{e_B}(\boldsymbol{w})}{\partial \boldsymbol{w_2}} & \cdots & \frac{\partial \boldsymbol{e_B}(\boldsymbol{w})}{\partial \boldsymbol{w_M}} \end{bmatrix}_{\boldsymbol{BXM}} \tag{3.64}$$

The $\boldsymbol{S}(\boldsymbol{w})$ is calculated as follows:

$$\boldsymbol{S}(\boldsymbol{w}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{B}} \boldsymbol{e_i}(\boldsymbol{w})\nabla^{\mathbf{2}}\boldsymbol{e_i}(\boldsymbol{w}) \tag{3.65}$$

Substituting (3.62) and (3.63) in (3.61); hence $\boldsymbol{\Delta w}$ could be written as follows.

$$\boldsymbol{\Delta w} = -[\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\mathbf{J}(\boldsymbol{w}) + \boldsymbol{S}(\boldsymbol{w})]^{-\mathbf{1}}\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\boldsymbol{e}(\boldsymbol{w}) \tag{3.66}$$

The calculations of the second-order partial derivatives in the term $\boldsymbol{S}(\boldsymbol{w})$ are computationally high. Hence, according to the Gauss-Newton method, the value of $\boldsymbol{S}(\boldsymbol{w}) \approx \mathbf{0}$. Based on that, the calculation of the Hessian matrix is approximated as following $\nabla^{\mathbf{2}}\boldsymbol{L_{CNN}}(\boldsymbol{w}) \approx \mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\mathbf{J}(\boldsymbol{w})$. The weights update ($\boldsymbol{\Delta w}$) could be approximated as follows:

$$\boldsymbol{\Delta w} \approx -[\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\mathbf{J}(\boldsymbol{w})]^{-\mathbf{1}}\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\boldsymbol{e}(\boldsymbol{w}) \tag{3.67}$$

The LMB is a modified version of the Gauss-Newton method where it approximates the calculations of the Hessian matrix $\nabla^{\mathbf{2}}\boldsymbol{L_{CNN}}(\boldsymbol{w})$ by not neglecting the $\boldsymbol{S}(\boldsymbol{w})$. It compensates for the term $\boldsymbol{S}(\boldsymbol{w})$ $by\ \zeta \boldsymbol{I_{MXM}}$, where ζ is the damping factor/adaptive learning rate, and $\boldsymbol{I}$ is the identity matrix. Hence, the Hessian matrix in this case, is approximated as follows.

$$\nabla^{\mathbf{2}}\boldsymbol{L_{CNN}}(\boldsymbol{w}) \approx \mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\mathbf{J}(\boldsymbol{w}) + \boldsymbol{\zeta I_{MXM}} \tag{3.68}$$

Substituting (3.62) and (3.68) in (3.61); thus, the weights update could be approximated as

$$\boldsymbol{\Delta w} \approx -[\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\mathbf{J}(\boldsymbol{w}) + \boldsymbol{\zeta I_{MXM}}]^{-\mathbf{1}}\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{w})\boldsymbol{e}(\boldsymbol{w}) \tag{3.69}$$

The damping factor ζ is multiplied by factor $\rho > 1$. Typically, the initial values are $\zeta = 0.001,\ \rho = 10$, when an iteration of training leads to an increased loss function. When ζ gets larger, it behaves more like the steepest gradient descent (with a step $1/\zeta$) and is helpful when the current solution is far from the optimal one. In such cases, a

larger ζ value encourages smaller weight updates and prevents the algorithm from making huge steps that might lead to divergence. When an iteration leads to a decreased $\boldsymbol{L_{CNN}(w)}$, then ζ is divided by ρ. When ζ gets smaller (close to zero), the algorithm behaves more like the Gauss-Newton method, which is well-suited for converging quickly when the current solution is close to the optimal solution. In this case, it approximates the Hessian matrix using second-order derivatives. The intuition behind the LMB optimizer is depicted in Figure 3.8.

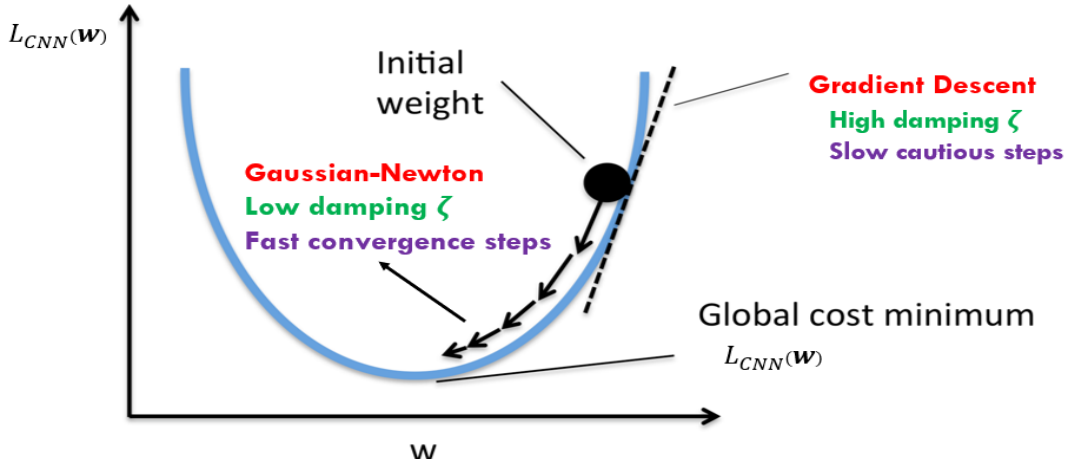


Figure 3.8 Intuition behind LMB optimizer

ii. Backpropagation algorithm

The cornerstone of the LMB algorithm is the calculation of the Jacobian matrix. This is done by calculating the partial derivative of each residual error at a certain sample $e_i(\boldsymbol{w})$ with respect to certain weight $w_{l-m}^{[l]}(q,r)$ in weight matrix $\boldsymbol{W_{l-m}^{l}}$ and with respect to certain bias $b_r^{[l]}$ in the bias vector $\boldsymbol{b}^{[l]}$, where (q,r) are the row and column numbers in the matrix.

$$\frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial w_{l-m}^{[l]}(q,r)}} \text{ , } \frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial b_r^{[l]}}} \tag{3.70}$$

The gradient is calculated using the backpropagation algorithm [102] through the derivative chain rule, which is applied to the cascaded-forward equations (3.40) to (3.49). The illustration of this process is a bit complicated, so let's illustrate the process starting from layer 5 in the network and ending up to layer 1 (backpropagation) to show the idea.

In layer 5, assume we want to find the partial derivatives of $e_i(\boldsymbol{w})$ with respect to to $w_{A_{l-m}}^{[5]}(q,r)$ and $b_r^{[5]}$ of the network. This can be calculated as follows.

$$\frac{\partial e_i(\boldsymbol{w})}{\partial w_{A_{l-m}}^{[5]}(q,r)} = \frac{\partial e_i(\boldsymbol{w})}{\partial \hat{u}_{cpc}^{(i)}(\boldsymbol{w})} * \frac{\partial \hat{u}_{cpc}^{(i)}(\boldsymbol{w})}{\partial \boldsymbol{Z}^{[5](i)}} * \frac{\partial \boldsymbol{Z}^{[5](i)}}{\partial w_{A_{l-m}}^{[5]}(q,r)} = \boldsymbol{dZ}^{[5](i)} * \frac{\partial \boldsymbol{Z}^{[5](i)}}{\partial w_{A_{l-m}}^{[5]}(q,r)} \tag{3.71}$$

For notation simplicity, consider the following

$$\boldsymbol{dZ^{[5](i)}} = \frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial \hat{u}_{cpc}^{(i)}(w)}} * \frac{\boldsymbol{\partial \hat{u}_{cpc}^{(i)}(w)}}{\boldsymbol{\partial Z^{[5](i)}}} \tag{3.72}$$

Thus,

$$\frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial b_r^{[5]}}} = \boldsymbol{dZ^{[5](i)}} * \frac{\boldsymbol{\partial Z^{[5](i)}}}{\boldsymbol{\partial b_r^{[5]}}} \tag{3.73}$$

In layer 4, the partial derivatives of $e_i(\boldsymbol{w})$ with respect to to $w_{A_{l-m}}^{[4]}(q,r)$ and $b_r^{[4]}$ are

$$\boldsymbol{dZ^{[4](i)}} = \boldsymbol{dZ^{[5](i)}} * \frac{\boldsymbol{\partial Z^{[5](i)}}}{\boldsymbol{\partial A^{[4](i)}}} * \frac{\boldsymbol{\partial A^{[4](i)}}}{\boldsymbol{\partial Z^{[4](i)}}} \tag{3.74}$$

$$\frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial w_{A_{l-m}}^{[4]}(q,r)}} = \boldsymbol{dZ^{[4](i)}} * \frac{\boldsymbol{\partial Z^{[4](i)}}}{\boldsymbol{\partial w_{A_{l-m}}^{[4]}(q,r)}} \tag{3.75}$$

$$\frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial b_r^{[4]}}} = \boldsymbol{dZ^{[4](i)}} * \frac{\boldsymbol{\partial Z^{[4](i)}}}{\boldsymbol{\partial b_r^{[4]}}} \tag{3.76}$$

In layer 3, the partial derivatives of $e_i(\boldsymbol{w})$ with respect to to $w_{A_{l-m}}^{[3]}(q,r)$ and $b_r^{[3]}$ are

$$\boldsymbol{dZ^{[3](i)}} = \boldsymbol{dZ^{[4](i)}} * \frac{\boldsymbol{\partial Z^{[4](i)}}}{\boldsymbol{\partial A^{[3](i)}}} * \frac{\boldsymbol{\partial A^{[3](i)}}}{\boldsymbol{\partial Z^{[3](i)}}} + \boldsymbol{dZ^{[5](i)}} * \frac{\boldsymbol{\partial Z^{[5](i)}}}{\boldsymbol{\partial A^{[3](i)}}} * \frac{\boldsymbol{\partial A^{[3](i)}}}{\boldsymbol{\partial Z^{[3](i)}}} \tag{3.77}$$

$$\frac{\partial e_i(\boldsymbol{w})}{\partial w_{A_{l-m}}^{[3]}(q,r)} = \boldsymbol{dZ^{[3](i)}} * \frac{\partial \boldsymbol{Z^{[3](i)}}}{\partial w_{A_{l-m}}^{[3]}(q,r)} \tag{3.78}$$

$$\frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial b_r^{[3]}}} = \boldsymbol{dZ^{[3](i)}} * \frac{\boldsymbol{\partial Z^{[3](i)}}}{\boldsymbol{\partial b_r^{[3]}}} \tag{3.79}$$

In layer 2, the partial derivatives of $e_i(\boldsymbol{w})$ with respect to to $w_{A_{l-m}}^{[2]}(q,r)$ and $b_r^{[2]}$ are

$$\boldsymbol{dZ^{[2](i)}} = \boldsymbol{dZ^{[3](i)}} * \frac{\boldsymbol{\partial Z^{[3](i)}}}{\boldsymbol{\partial A^{[2](i)}}} * \frac{\boldsymbol{\partial A^{[2](i)}}}{\boldsymbol{\partial Z^{[2](i)}}} + \boldsymbol{dZ^{[4](i)}} * \frac{\boldsymbol{\partial Z^{[4](i)}}}{\boldsymbol{\partial A^{[2](i)}}} * \frac{\boldsymbol{\partial A^{[2](i)}}}{\boldsymbol{\partial Z^{[2](i)}}} + \boldsymbol{dZ^{[5](i)}} * \frac{\boldsymbol{\partial Z^{[5](i)}}}{\boldsymbol{\partial A^{[2](i)}}} * \frac{\boldsymbol{\partial A^{[2](i)}}}{\boldsymbol{\partial Z^{[2](i)}}} \tag{3.80}$$

$$\frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial w_{A_{l-m}}^{[2]}(q,r)}} = \boldsymbol{dZ^{[2](i)}} * \frac{\boldsymbol{\partial Z^{[2](i)}}}{\boldsymbol{\partial w_{A_{l-m}}^{[2]}(q,r)}} \tag{3.81}$$

$$\frac{\boldsymbol{\partial e_i(w)}}{\boldsymbol{\partial b_r^{[2]}}} = \boldsymbol{dZ^{[2](i)}} * \frac{\boldsymbol{\partial Z^{[2](i)}}}{\boldsymbol{\partial b_r^{[2]}}} \tag{3.82}$$

In layer 1, the partial derivatives of $e_i(\boldsymbol{w})$ with respect to to $w_X^{[1]}(q,r)$ and $b_r^{[1]}$ are

$$\boldsymbol{dZ^{[1](i)}} = \boldsymbol{dZ^{[2](i)}} * \frac{\boldsymbol{\partial Z^{[2](i)}}}{\boldsymbol{\partial A^{[1](i)}}} * \frac{\boldsymbol{\partial A^{[1](i)}}}{\boldsymbol{\partial Z^{[1](i)}}} + \boldsymbol{dZ^{[3](i)}} * \frac{\boldsymbol{\partial Z^{[3](i)}}}{\boldsymbol{\partial A^{[1](i)}}} * \frac{\boldsymbol{\partial A^{[1](i)}}}{\boldsymbol{\partial Z^{[1](i)}}} + \boldsymbol{dZ^{[4](i)}} * \frac{\boldsymbol{\partial Z^{[4](i)}}}{\boldsymbol{\partial A^{[1](i)}}} * \frac{\boldsymbol{\partial A^{[1](i)}}}{\boldsymbol{\partial Z^{[1](i)}}} + \boldsymbol{Z^{[5](i)}} * \frac{\boldsymbol{\partial Z^{[5](i)}}}{\boldsymbol{\partial A^{[1](i)}}} * \frac{\boldsymbol{\partial A^{[1](i)}}}{\boldsymbol{\partial Z^{[1](i)}}} \tag{3.83}$$

$$\frac{\partial e_i(w)}{\partial w_X^{[1]}(q,r)} = dZ^{[2](i)} * \frac{\partial Z^{[1](i)}}{\partial w_X^{[1]}(q,r)} \tag{3.84}$$

$$\frac{\partial e_i(w)}{\partial b_r^{[1]}} = dZ^{[1](i)} * \frac{\partial Z^{[1](i)}}{\partial b_r^{[1]}} \tag{3.85}$$

iii. The summary of the optimization process

The following steps show the summary of applying the LMB algorithm

1- Feed B samples of the training input feature matrix $\boldsymbol{X}$ forward to the network and compute the estimated output vector $\widehat{\boldsymbol{u}}_{\boldsymbol{cpc}}$ based on equations (3.40) to (3.49).
2- At step k, Compute the residual error vector $\boldsymbol{e}(\boldsymbol{w_k})$ using equation (3.58), then use it to calculate the sum of square errors loss function $L_{CNN}(\boldsymbol{w_k})$ using equation (3.56).
3- Compute the partial derivative terms in the Jacobian matrix (3.64) using the backpropagation algorithm.
4- Solve the equation (3.69) to get the network updates $(\boldsymbol{\Delta w_k})$
5- Repeat steps (1) and (2) to compute the next step loss function $L_{CNN}(\boldsymbol{w_k} + \boldsymbol{\Delta w_k})$
   a. If the $L_{CNN}(\boldsymbol{w_k} + \boldsymbol{\Delta w_k}) < L_{CNN}(\boldsymbol{w_k}) \rightarrow$ reduce the damping factor ζ by dividing over ρ and update the network parameters $\boldsymbol{w_{k+1}} = \boldsymbol{w_k} + \boldsymbol{\Delta w_k}$.
   b. Else if $L_{CNN}(\boldsymbol{w_k} + \boldsymbol{\Delta w_k}) > L_{CNN}(\boldsymbol{w_k}) \rightarrow$ increase ζ by multiplying with ρ and resolve equation (3.69) to update $(\boldsymbol{\Delta w_k})$.
6- The training is assumed to have converged when one of the following conditions has been reached (training stopping criteria):
   a. A maximum predetermined number of training epochs
   b. A minimum predetermined value of the loss function
   c. A minimum predetermined value of the norm of the gradient in (3.62)
   d. A maximum predetermined value of ζ (referring to the divergence of the algorithm)
   e. A predetermined number of validation checks (to avoid overfitting problem)

c) The CNN-C Training Results

The training of the CNN-C has been passed through four main stages: optimization, validation, hyperparameters tunning, and testing. This Subsection handles the results from the training and testing of the neural networks under different conditions.

Table 3.6 shows the training results of multi-configured CNN, where the number of layers and neurons are the hyperparameters to be tuned after each training process. The predetermined values of the network stopping criteria are mentioned in the top head of each column in Table 3.6.

Table 3.6 Training results of different CNN topologies

Metrics (Target) / Network Configuration	Epochs (1000)	Training Time (hrs:mins: secs)	Maximum Validation Check (12)	Norm gradient value $(1e-7)$	ζ Value $(1e10)$	Training loss function value $(1e-5)$	Testing loss function value
Two hidden layers [7-5] neurons	589	00:02:23	12	0.0422	1e-06	6.525e-4	1.602e-4
Two hidden layers [10-7] neurons	319	00:01:39	12	0.00651	1e-05	4.771e-4	1.217e-4
Three hidden layers [7-5-3] neurons	379	00:01:58	12	0.0679	1e-07	2.265e-4	5.813e-5
Three hidden layers [15-10-5] neurons	156	00:02:23	12	0.052	1e-06	2.256e-4	6.96e-5
Four hidden layers [10-7-5-3]	600	00:07:06	12	0.00454	1e-06	7.829e-5	3.819e-5
Four hidden layers [15-10-7-5]	208	00:06:30	12	0.00149	1e-05	9.49e-05	6.718e-5
Four hidden layers [20-15-10-5]	108	00:08:36	12	0.00644	1e-07	3.09e-05	1.7e-5

The training process is conducted on personal laptop (Lenovo Y50-70) with the following specifications:

- Processor: Intel® Core ™ i7-4720 HQ (4 cores/ 8 threads) @ 2.60 GHZ
- Installed Memory: 16 GB
- Graphic Processing Unit (GPU): Nvidia GTX 960m (4GB memory)
- Disk drive: Samsung SSD 870 EVO 500 GB
- Operating System: Windows 10

As shown in Figure 3.9, the training progress of the CNN is enhanced over the epochs. The training loss function value decreases after each epoch. The training is stopped when the validation loss value is kept constant or does not enhance anymore after a certain number of epochs (12 in our case). The stopping criterion is called “early stopping.” It helps the model to avoid overfitting problems. The CNN model gets overfitted when performing well on training data but not well on validation or testing data.

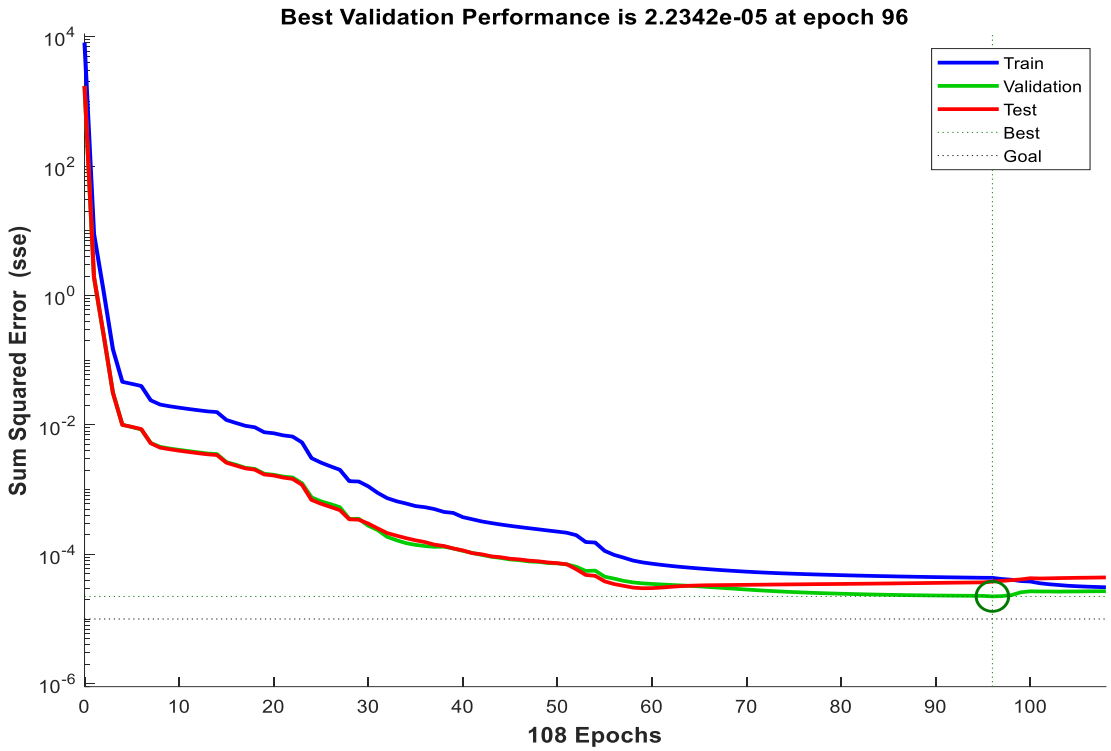


Figure 3.9 The learning curve of the CNN showing the sum of squared error at each epoch

3.4. Proposed F-DDPG Controller Design

The proposed novel F-DDPG controller is designed to regulate the wind turbine system in region 3. The training and deployment phases of the controller are represented as block diagrams in Subsection 3.4.1. Following that, the mathematics of the training algorithm are explained in Subsection 3.4.2. The topologies of actor and critic neural networks are presented in Subsections 3.4.3 and 3.4.4, respectively. The choice of the reward function is justified in Subsection 3.4.5. Finally, the implementation of the fuzzy logic with the multi-trained agents is conducted in Subsection 3.4.6.

3.4.1. Proposed controller structure

The application of the proposed controller has two phases: the training and deployment phases. In the training phase, six distinct DDPG agents are intended for training. Each agent is trained individually at a certain average stochastic wind speed operating point to cover all possible wind speeds operating points (14, 16, 18, 20, 22, 24) m/s in region 3. As shown in Figure 3.10, each agent comprises six main blocks: FMPC, experience replay buffer, online actor network, online critic network, target actor network, and target critic network.

Imitation learning is initially used to provide efficient samples to fasten training convergence. FMPC acts as the demonstrator to guide the DDPG agents. During the first five training episodes, the selector switch is pointed to the FMPC block to provide the control action to the environment. The selection of five initial episodes for efficient

sample collection is made empirically through trial and error. Testing with 1 to 4 initial episodes yielded unsatisfactory results.

The experience replay buffer R_B block serves as a memory storage for past experiences. The buffer contains tuples, each of which comprises the current state (s_t), the current action output (a_t), the observed reward (r_t), and the subsequent state (s_{t+1}) following the execution of a_t at time step t. Initially, FMPC supplies efficient samples to the experience replay buffer block.

The online actor network block is parameterized by ϕ^{μ} weights and it is responsible for outputting specific (deterministic) pitch control action $\mu(s|\phi^{\mu})$ given the states (s) of the environment as inputs. Ornstein-Uhlenbeck random noise, represented in equations (3.88) to (3.90), is added to the actor network output. It allows the agent to explore the action space more coherently due to temporal correlation, leading to better performance and faster learning [52].

The online critic network block is parameterized by ϕ^{Q} weights. It is used to evaluate the online actor network performance. This is done by estimating the value function based on the state-action pair as input. It outputs a single scalar value $Q(s, a|\phi^{Q})$ representing the estimated return (Q-value) of that state-action pair.

The target actor network block aims to mitigate the issues of data correlation and non-stationarity, improving the algorithm's ability to converge to a good policy [52]. It is parameterized by $\phi^{\mu'}$ weights and it serves as a more stable approximation of the policy represented by the online actor network. It is used to generate the target action $\mu'(s_{x+K}|\phi^{\mu'})$ based on the next K-step states s_{x+K}.

The target critic network block aims to mitigate the risks of policy oscillation and divergence, making the learning process more robust and stable. It is parameterized by $\phi^{Q'}$ and it is used to output the target estimated return $Q'(s_{x+K}, \mu'(s_{x+K}|\phi^{\mu'})|\phi^{Q'})$ based on the target action and next K-step states. Then, the K-step target Q-value y_x is computed using the discounted sum of the target estimated return and the actual rewards received ($r_x, r_{x+1}, \dots, r_{x+k-1}$). The number K refers to the number of time steps into the future that the algorithm considers when calculating y_x. K-step target return often balances the bias-variance tradeoff. A 1-step return is unbiased but may have high variance, while a return that considers the entire future (i.e., until the end of the episode) has low variance but may be biased [57]. K-step returns represent a middle choice which yield faster and more stable learning.

The learning process of each agent is done iteratively based on predefined numbers of episodes (E), where each episode consists of predefined numbers of timesteps (T). The learning process continuously updates the parameters of the online actor-critic networks and target actor-critic networks at each timestep (t). The online actor network parameters are updated to produce actions that maximize the objective function (J_t) representing the cumulative expected return, as approximated by the online critic network. The online critic network parameters are updated to minimize the mean-square-error (MSE) loss function (L_t) which represents the difference between estimated and target Q-values. The updating for online actor-critic networks parameters is done by using the adaptive moment estimation (ADAM) optimization technique. It is

explicitly chosen because of the randomness in the optimization process due to the random mini-batched data used. Also, it is computationally efficient, needs fewer memory requirements, and is a suitable choice for stochastic optimization [99]. The target actor-critic networks parameters are softly updated to track the online actor-critic networks, making the learning process more stable.

The control signal flow shown in the top left of Figure 3.10 is demonstrated as steps. The initial step (zero step) is to initialize the experience replay buffer memory and online actor-critic networks weights, which are then passed to initialize the target actor-critic networks.

Steps 1 through 5 are dedicated to the sequence followed to store tuples in R_B. In first step, the agent receives the current states s_t (θ_{DT} , $\Delta\omega_{rotor}$, and ω_{DT}) from the environment and passes it to the FMPC and the online actor network. The selector switch is pointed to FMPC in the initial five learning episodes. After the five episodes are done, the selector switch is pointed to the online actor network to take control. In the second step, the control action a_t, whether exerted from FMPC or actor network, is passed to the environment and executed. Also, the reward r_t is calculated based on the normalized absolute error of rotor speed $\left|\left|\Delta\omega_{rotor}\right|\right|$ and the normalized absolute error of generator output power $\left|\left|\Delta P_{Gen}\right|\right|$, as mentioned in (3.87). In the third step, the next states s_{t+1} are observed based on the a_t executed in step 2. In the fourth step, the transition tuple (s_t, a_t, r_t, s_{t+1}) is stored in the R_B. In the fifth step, random M-numbered tuples are selected to form mini-batch samples which are used to train online actor-critic networks.

Steps 6 through 9 are dedicated to updating online critic network parameters. In step six, s_{x+k} is passed to the target actor network. The objective of step seven is to prepare the inputs for the online critic network and target critic network. $\mu'\left(s_{x+K}\middle|\phi^{\mu'}\right)$ and s_{x+k} are passed to the target critic network. The state-action pair (s_x, a_x) is passed to the online critic network. The objective of step eight is to construct the MSE loss function L_t. The output $Q(s_x, a_x|\phi^Q)$ from the online critic network, the output from the target critic network $Q'\left(s_{x+K}, \mu'\left(s_{x+K}\middle|\phi^{\mu'}\right)\middle|\phi^{Q'}\right)$, the actual rewards sequence $(r_x, r_{x+1} \ldots r_{x+k-1})$, and the previous tunned parameters of the critic network ϕ_{t-1}^Q are used to construct L_t as shown in equations (3.92) and (3.93). The objective in step nine is to minimize L_t through calculating gradient descent ∇L_t with respect to online critic network parameters ϕ^Q as shown in equation (3.94). To calculate the newly updated parameters ϕ_t^Q based on ∇L_t, ADAM optimization technique is used.

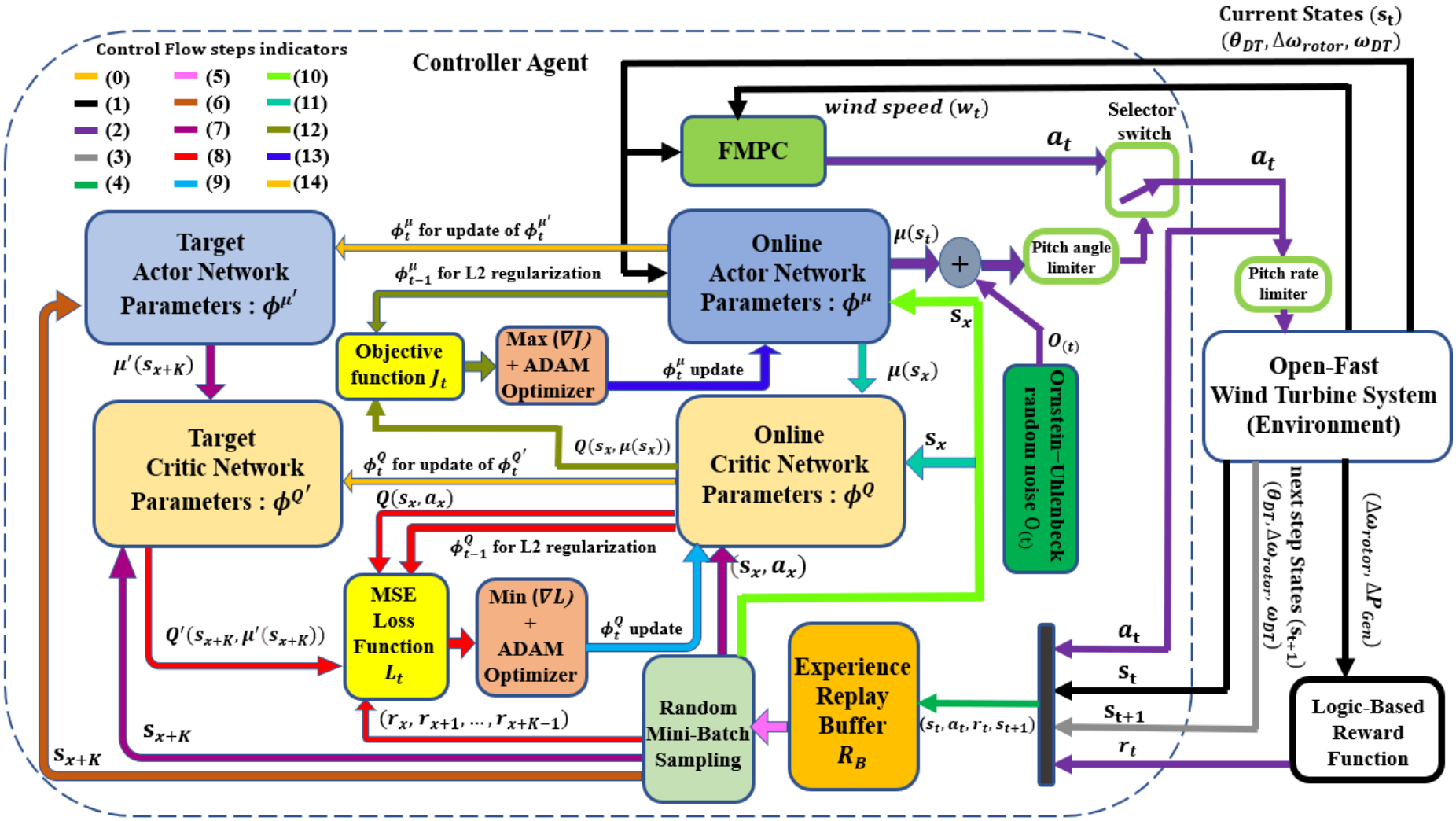


Figure 3.10 The training stage of F-DDPG controller

Steps 10 through 14 are dedicated to updating online actor network parameters. In step 10, the state s_x is passed to the online actor network. In step 11, the output control action $\mu(s_x)$ and s_x are passed to the online critic network to output the estimated Q-values $Q\big(s_x, \mu(s_x)\big)$. In step 12, the objective function J_t is constructed using $Q\big(s_x, \mu(s_x)\big)$ and previous tunned parameters of actor network ϕ_{t-1}^{μ} as mentioned in equation (3.98). The objective of step 13 is to maximize J_t through calculating its gradient with respect to online actor network parameters ϕ^{μ}, then to calculate the newly updated parameters ϕ_t^{μ} using ADAM optimization technique. In the final step (14th step), the target actor-critic networks parameters ($\phi_t^{Q'}, \phi_t^{\mu'}$) are smoothly updated based on the recent values of online actor-critic networks parameters (ϕ_t^{Q}, ϕ_t^{μ}) as mentioned in equations (3.100) and (3.101). The training sequence is repeated in the next iteration starting from step 1. Previously mentioned steps are explained mathematically in detail in Subsection 3.5.

After training (at the deployment stage), the agents' online actor networks are the only activated networks with already trained parameters that are no longer updated. The online actor network makes decisions without the need to learn from new reward values. Its primary focus in this stage is to exploit the output, not to explore the environment anymore. As a result, Ornstein-Uhlenbeck noise is not added to the actor network output. Each agent outputs the optimal pitch control action based on the current states, as shown in Figure 3.11. Then, the agents' outputs are processed by a fuzzy system, as explained in detail in Subsection 3.6. This results in the generation of the fuzzy optimal control action C_f which is applied to the environment.

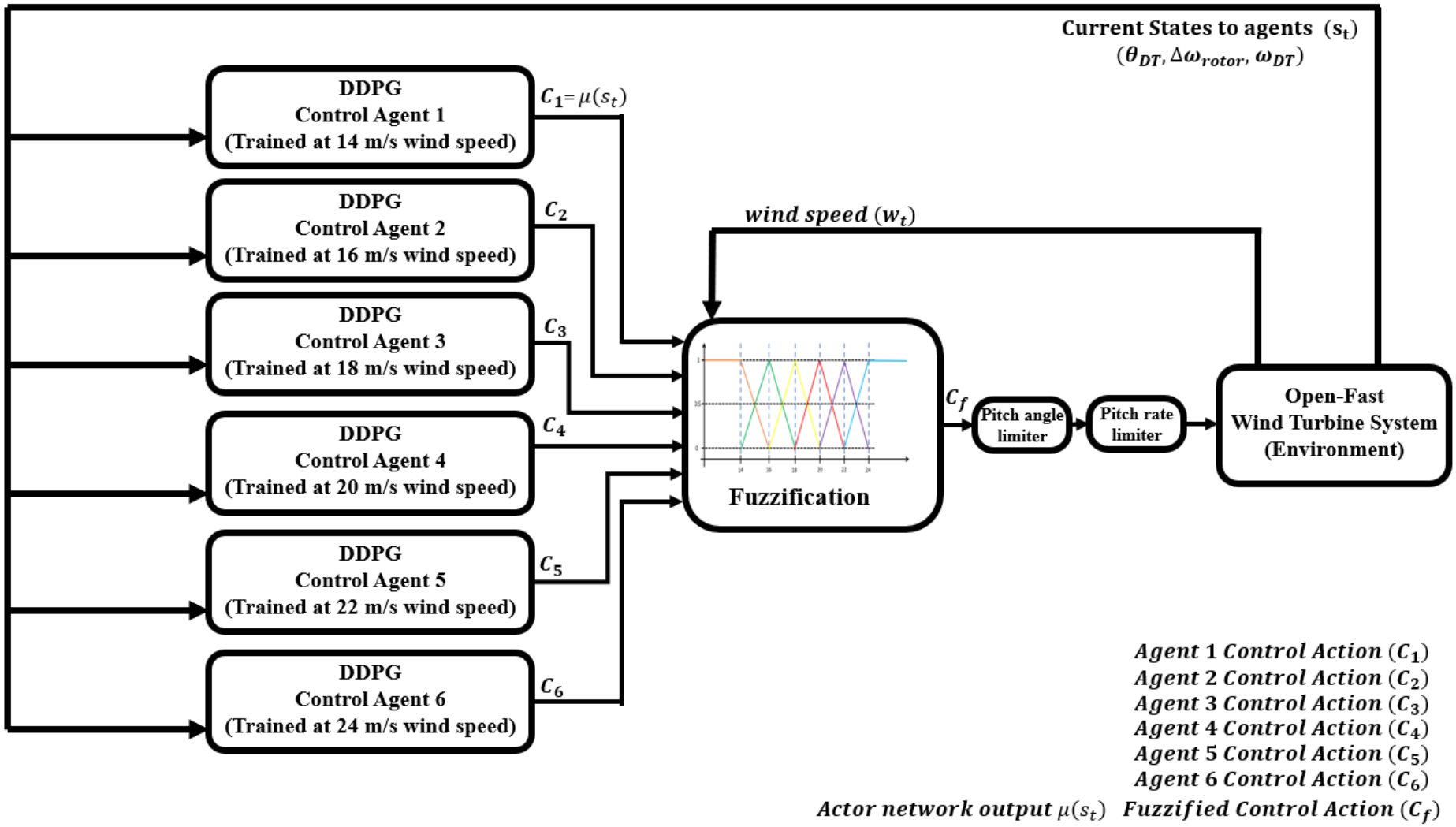


Figure 3.11 The deployment stage of the F-DDPG controller

3.4.2. F-DDPG Training Algorithm

Algorithm 5 explains the training algorithm of each DDPG control agent using imitation learning in the initial learning episodes. Imitation learning is done by collecting efficient samples from a fuzzy model predictive controller to guide and fasten the training process. The parameters and hyperparameters values of the training scenarios are selected based on various trials, as shown in Table 3.7. The training process for each agent is completed once all episodes have been iterated. Subsequently, at the deployment phase, the control actions of the six trained DDPG agents are processed by a fuzzy system using the method described in Subsection 3.6.

Algorithm 5 Deep deterministic policy gradient training enhanced by imitation learning and k-step bootstrapping.

- Online critic and actor networks weights (ϕ^Q, ϕ^μ) are initialized randomly using the Xavier-Initialization [103] :

$$\phi_j^Q \; or \; \phi_j^\mu \sim U\left[-\frac{\sqrt{6}}{\sqrt{N_j+N_{j+1}}}, \frac{\sqrt{6}}{\sqrt{N_j+N_{j+1}}}\right] \tag{3.86}$$

where ϕ_j^Q , ϕ_j^μ are the critic and actor networks parameters in certain layer j, respectively. N_j is the number of neurons in layer j, N_{j+1} is the number of neurons in the next layer, U is a uniform distribution

- The critic and actor target networks weights ($\phi^{Q'}, \phi^{\mu'}$) are initialized with the same online critic and actor networks weights.
- Empty experience replay buffer R_B is initialized with length 10^6 samples as suggested in [52].
- Initialization of first $\left(f_{m_{(0)}}\right)$ and second $\left(s_{m_{(0)}}\right)$ moments values of ADAM optimization with zeros [99].
- Set the steps to look ahead K = 10.
- for episode number = 1: E, do
  - Receiving initial observation states tuple s_1
  - If (episode number < 5) do
    - for t (step) = 1: T do
    - exert output pitch control action a_t based on the FMPC.
    - execute pitch control action a_t, receive the next step states s_{t+1} and receive the reward r_t from the environment. Whereas the logic-based reward function is defined as:

$$r_t = \begin{cases} 20, \; if \; \left(\left||\Delta\omega_{rotor}|\right| < 0.3\% \; and \; \left||\Delta P_{Gen}|\right| < 0.3\%\right) \\ 10, \; else \; if \; \left(\left||\Delta\omega_{rotor}|\right| < 0.3\% \; or \; \left||\Delta P_{Gen}|\right| < 0.3\%\right) \\ 0, \; otherwise \\ where \;\; \left||\Delta P_{Gen}|\right| = \frac{P_t - P_{rated}}{P_{rated}} \; and \; \left||\Delta\omega_{rotor}|\right| = \frac{\omega_t - \omega_{rated}}{\omega_{rated}} \end{cases} \tag{3.87}$$

      Where the choice of 20, 10, 0 values are based on fine tuning.

    - Store (s_t, a_t, r_t, s_{t+1}) transition in R_B for purpose of collecting guided samples.

- Else → for t = 1: T do
  - For action exploration, Ornstein-Uhlenbeck random noise process formula is used:

$$O_{(t)} = O_{(t-1)} + \theta * \left(\xi - O_{(t-1)}\right) * T_s + \sigma_{(t)} * W_{(t)} \quad (3.88)$$

$$\sigma_\delta = 1 - 10^{\frac{log_{10}(0.5)}{HLT}} \rightarrow \sigma_{decayed} = \sigma_{(t)} * (1 - \sigma_\delta) \quad (3.89)$$

$$\sigma_{(t+1)} = max\left(\sigma_{decayed}, \sigma_{min}\right) \quad (3.90)$$

Where $O_{(t-1)}$ is the noise value at previous step, ξ is the noise mean value, θ is the mean attraction constant, T_s is the sampling interval, $\sigma_{(t)}$ noise standard deviation, $W_{(t)}$ is wiener process, σ_δ is constant value representing the standard deviation decay rate , HLT is the half life time for noise standard deviation (as number of samples) , $\sigma_{decayed}$ is decayed noise standard deviation, $\sigma_{(t+1)}$ next-step noise standard deviation, σ_{min} is the minimum value of noise standard deviation. The suggested values for the parameters are shown in Table 3.7.

  - Initialization of Ornstein-Uhlenbeck random noise process parameters is created only at the first timestep ($t_0 = 1$) and it is mentioned in Table 3.7.
  - exert output pitch control action, based on random noise and actor policy at the instant timestep:

$$a_t = clip(\ \mu(s_t|\phi^\mu) + \ O_{(t)},\ a_{min},\ a_{max}) \quad (3.91)$$

    Where $\mu(s_t|\phi^\mu)$ is the action taken at the current state s_t as per the online actor network with parameters ϕ^μ. a_{min} and a_{max} are the minimum and maximum permissible pitch angles.
  - execute pitch control action a_t, receive next step states s_{t+1} and receive the reward r_t from the environment.
  - Store (s_t, a_t, r_t, s_{t+1}) transition in R_B.
  - Transitions tuples are sampled randomly from R_B based on B number of minibatch with at least K-consecutive-steps sequence of transitions i.e., zero transition (s_x, a_x, r_x, s_{x+1}), first transition $(s_{x+1}, a_{x+1}, r_{x+1}, s_{x+2})$, … K-transition $(s_{x+K}, a_{x+K}, r_{x+K}, s_{x+K+1})$
  - Calculate the K-step Q-target return (y_x) (target expected future reward) which considers a sum of discounted future rewards plus the value of the future state-action pair estimated by the target critic network. For a given sample x^{th} in the mini-batch, y_x could be formulated as:

$$y_x = r_x + \gamma r_{x+1} + \ \gamma^2 r_{x+2} + \cdots + \gamma^{K-1} r_{x+K-1} + \gamma^K Q'\left(s_{x+K}, \mu'\left(s_{x+K}\middle|\phi^{\mu'}\right)\middle|\phi^{Q'}\right) \quad (3.92)$$

    where $r_x, r_{x+1}, \ldots, r_{x+K-1}$ are the actual rewards at each step from x to $x + K - 1$, γ is the discount factor, $\mu'\left(s_{x+K}\middle|\phi^{\mu'}\right)$ is the action taken at the state s_{x+K} as per the target actor network with parameters $\phi^{\mu'}$.

$Q'\left(s_{x+K}, \mu'\left(s_{x+K}\middle|\phi^{\mu'}\right)\middle|\phi^{Q'}\right)$ is the Q-value of taking that action at s_{x+K} according to the target critic network with parameters $\phi^{Q'}$.

- The loss function for the critic network is formulated by employing the MSE as the metric for evaluating the discrepancy between predicted and target values, and incorporating L2 regularization to prevent overfitting and enhance model generalization:

$$L_t\left(\phi_{ij}^{Q}\right) = \frac{1}{B}\left(\sum_{x=1}^{B}\left(Q\left(s_x, a_x\middle|\phi_{ij}^{Q}\right) - y_x\right)^2\right) + \lambda_Q \sum_{i=action\ path}^{main\ path} \sum_{j=1}^{PP} ||\phi_{ij}^{Q}||^2 \quad (3.93)$$

Where B is the number of minibatch samples, PP is the number of layers in each path, λ_Q is the L2 regularization factor, $Q\left(s_x, a_x\middle|\phi_{ij}^{Q}\right)$ is the estimated Q-value of taking action a_x at state s_x as per online critic network.

- Calculate the gradient descent of the loss function $L\left(\phi_{ij}^{Q}\right)$ with respect to the parameters $\left(\phi_{ij}^{Q}\right)$ for minimization purpose:

$$\nabla_{\phi_{ij}^{Q}} L_t = \left(\frac{2}{B}\right)\left(\sum_{x=1}^{B}\left(Q\left(s_x, a_x\middle|\phi_{ij}^{Q}\right) - y_x\right) * \nabla_{\phi_{ij}^{Q}} Q\left(s_x, a_x\middle|\phi_{ij}^{Q}\right)\right) + 2 * \lambda_Q * \left|\left|\phi_{ij}^{Q}\right|\right| \quad (3.94)$$

- critic network parameters (ϕ^{Q}) are updated using ADAM optimization technique:
  - update the first $f_{m_{\phi^Q}}$ and second $s_{m_{\phi^Q}}$ moments estimate then correct the bias:

$$f_{m(t)_{\phi^Q}} = \beta_1 f_{m(t-1)_{\phi^Q}} + (1-\beta_1) * (\nabla_{\phi_{ij}^{Q}} L_t) \rightarrow \hat{f}_{m(t)_{\phi^Q}} = \frac{f_{m(t)_{\phi^Q}}}{(1-\beta_1^t)} \quad (3.95)$$

$$s_{m(t)_{\phi^Q}} = \beta_2 s_{m(t-1)_{\phi^Q}} + (1-\beta_2) * \left(\nabla_{\phi_{ij}^{Q}} L_t\right)^2 \rightarrow \hat{s}_{m(t)_{\phi^Q}} = \frac{s_{m(t)_{\phi^Q}}}{(1-\beta_2^t)} \quad (3.96)$$

Where β_1, β_2 are the exponential decay rates for first and second moment estimates, respectively. $f_{m(t)_{\phi^Q}}$, $s_{m(t)_{\phi^Q}}$ are the first and second moment estimates of the critic network parameters. $\hat{f}_{m(t)_{\phi^Q}}, \hat{s}_{m(t)_{\phi^Q}}$ are the corrected first and second moments respectively.

  - update the parameters ϕ^{Q} for each layer as following:

$$\phi_t^Q = \phi_{t-1}^Q - \alpha_Q \left(\frac{\hat{f}_{m(t)_{\phi^Q}}}{\sqrt{\hat{s}_{m(t)_{\phi^Q}}} + \delta} \right) \quad (3.97)$$

where α_Q is the learning rate of critic network, δ is a numerical factor to avoid division by zero.

- The objective function J_t which represents the cumulative reward for the actor network parameterized by ϕ^μ is defined as:

$$J_t\left(\mu\left(s_x\middle|\phi_j^\mu\right)\right) = \left(\frac{1}{B}\right)\sum_{x=1}^{B} Q\left(s_x, \mu\left(s_x|\phi_j^\mu\right)|\phi^Q\right) + \lambda_\mu \sum_{j=1}^{U_A} ||\phi_j^\mu||^2 \quad (3.98)$$

Where U_A is the total number of layers in actor policy network, $\mu(s_x|\phi^\mu)$ is the action taken at the state s_x as per the online actor network with parameters ϕ_j^μ, $Q\left(s_x, \mu\left(s_x|\phi_j^\mu\right)|\phi^Q\right)$ is the estimated Q-value of taking that action at state s_x according to the online critic network with parameters ϕ^Q. λ_μ is the L2 regularization factor of actor network to avoid overfitting.

- Calculate the gradient ascent of the objective function $J_t\left(\mu\left(s_x\middle|\phi_j^\mu\right)\right)$ with respect to the parameters $\left(\phi_j^\mu\right)$ based on chain-rule for maximization purpose:

$$\nabla_{\phi_j^\mu} J_t = \left(\frac{1}{B}\right)\sum_{x=1}^{B} \nabla_{\mu(s_x)} Q(s_x, \mu(s_x|\phi^\mu)|\phi^Q) \nabla_{\phi_j^\mu} \mu(s_x|\phi^\mu) + 2 * \lambda_\mu * \left|\left|\phi_j^\mu\right|\right| \quad (3.99)$$

- Actor network parameters ϕ^μ for each layer are updated using ADAM optimization technique.
- Smooth update for both actor and critic target networks parameters $(\phi_t^{\mu'}, \phi_t^{Q'})$ respectively using smoothing factor τ:

$$\phi_t^{Q'} = \tau\phi_t^Q + (1-\tau)\phi_{t-1}^{Q'} \quad (3.100)$$

$$\phi_t^{\mu'} = \tau\phi_t^\mu + (1-\tau)\phi_{t-1}^{\mu'} \quad (3.101)$$

Table 3.7 Parameters and hyperparameters for F-DDPG agents training

Parameters	Symbol	Value
Number of episodes	E	300
Time steps per episode	T	8000
Sampling interval	T_s	0.0125
Time step	t	0.0125
Minimum permissible pitch angle	a_{min}	0 rad
Maximum permissible pitch angle	a_{max}	$\pi/2$ rad
Learning rate of actor network	α_μ	$1 * 10^{-4}$
Learning rate of critic network	α_Q	$1 * 10^{-3}$
L2 Regularization factor of actor network	λ_μ	0.1
L2 Regularization factor of critic network	λ_Q	0.01
Discount factor	γ	0.95
exponential decay rate for 1st moment estimates	β_1	0.9
exponential decay rate for 2nd moment estimates	β_2	0.999
Minibatch size	B	128
Target network smooth factor	τ	0.005
Experience buffer samples size	R_B	10^6
Initial noise value = Initial action value	$O_{(t_0=1)}$	0.3
Noise process mean	ξ	0
Noise model mean attraction constant	θ	0.15
Initial noise standard deviation	$\sigma_{(t_0)}$	$\frac{0.01*(A_{max}-A_{min})}{\sqrt{T_s}} = 0.14$
Half-life-time Samples	HLT	4000
Noise standard deviation decay rate	σ_δ	$1.7327 * 10^{-4}$
Minimum standard deviation	σ_{min}	0
K-step lookahead	K	10
Random number (with mean = 0, standard deviation =1)	$R(t)$	Auto − generated
Wiener process	$W_{(t)}$	$R(t) * \sqrt{T_s}$ = Auto − generated

3.4.3. Actor networks topology

This network takes in the current state of the environment and outputs the pitch control action. Training the online actor network aims to map states to actions to maximize the expected cumulative reward. This is done through chain-rule backpropagation derivative rules, as shown briefly in Figure 3.12. The aim of training the target actor network is to have stable training by slowly updating target network parameters instead of quickly changing its parameters. Hence, the policy learning process does not become unstable or divergent. The slow updating of the target network allows for fast convergence and mitigates the oscillations and divergence in the learning process because neural networks are prone to biasing and instability [52].

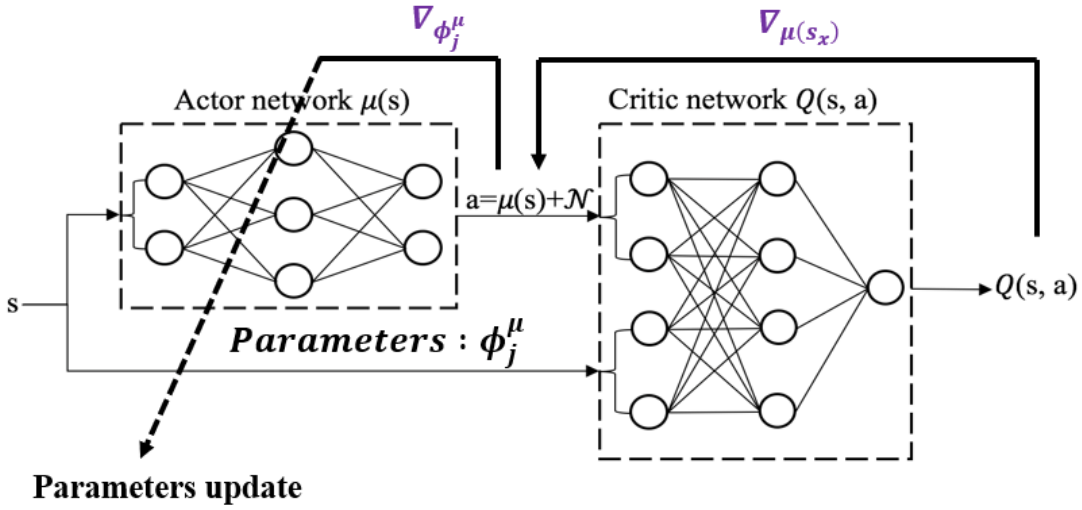


Figure 3.12 Overview on actor-network optimization scheme

The choice of actor network layers and neurons has been initially based on two hidden series of fully connected layers with 400 and 300 neurons, respectively, as mentioned in [52]. The training results have not been satisfactory due to the high complexity of the relation between the observed states and the output control action. The number of layers and neurons in each has been a crucial hyperparameter to tune. So, further modifications have been made to choose a balanced number of layers to extract the complex patterns between input states and output and avoid the trap of overfitting if a specific scenario is repeated accidentally during the online training. The empirical training findings were significant in selecting four series layers among various trials of two, three, and four series layers.

The suggested architecture, shown in Figure 3.13, is for both online and target actor networks. The architecture consists of a feature input layer, followed by a series of four hidden fully connected and tanh activation layers, ending up with an output layer that exerts the pitch control action before adding the desired noise for the exploration purpose of the DDPG algorithm. The feature input layer is designed to accept three states as input, representing the system's current state. It is designed without normalization to preserve the original scale of the input data. Even though these states are partially informative as not all non-linearity states are included, they were enough to give significant results in training.

The proposed architecture applies to both the online and target actor networks. Each actor network starts with a feature input layer for the states, succeeded by four hidden layers with neuron counts of 125, 100, 75, and 50, in that order. A tanh activation function follows every hidden layer. The architecture ends with a single-neuron output layer representing the control action.

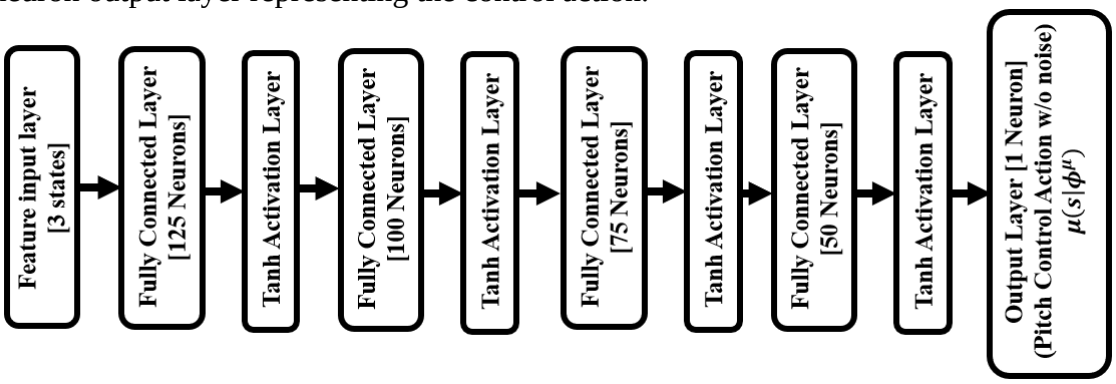


Figure 3.13 Neural network architecture of the actor network

The computational training time depends on several factors: the complexity of the model, the number of layers and neurons of a neural network, optimization technique, and batch size. Doubling the number of parameters in a neural network often doubles its computation training time [77]. According to the method of calculating the number of parameters in the fully connected network in [104], the estimated adjustable parameters for each actor network is 24,525 with four hidden layers. Using the topology in [52] and [60], each actor network would have 130,000 parameters with two hidden layers. According to our available processing capabilities, the computational training time of the suggested topology of the actor and critic neural networks has scored 375.78 seconds as an average total elapsed time per episode during the training process. In comparison, the topology stated in [52] and [60] has scored 602.79 seconds. Thus, our suggested topology has decreased the computational training time by a factor of 1.6.

The selection of activation functions in neural networks, specifically for hidden units, remains a key research area with few definitive guidelines [77]. While the rectified linear unit (ReLU) is often preferred [77], [96], it was not effective in our tests due to the "dying ReLU" issue, where negative inputs lead to zero outputs and vanishing gradients in deeper networks [105]. Given that the state inputs are bounded, we choose the tanh function, which scales inputs between -1 and 1. This maintains bounded neuron outputs without zero outputs and softens gradient vanishing issues [96]. Furthermore, the DDPG algorithm's performance is greatly influenced by the choice of hyperparameters [106], including the activation function. Thus, the tanh function has proven to stabilize the learning process and enable the network to capture more complex non-linear patterns in the data, underlining its empirical advantage for maintaining learning stability in our study.

The output layer of the network is a fully connected layer with one neuron, the same as the number of control actions, which is one in this case. This layer serves as the final step in the network, providing the continuous action values that the agent will provide to the environment based on the current state. The architecture of this actor network is carefully chosen based on trial and error to balance the complexity of the

model with the computational efficiency required for training and deployment in the F-DDPG algorithm.

3.4.4. Critic networks Topology

This network is structured to estimate the Q-value function, which measures the expected future rewards based on the agent's actions and environment states. The aim of training the parameters of this network is to understand how well the actor network takes control action based on the given states. As the critic network understanding of the system dynamics improves through training, the accuracy of the Q-value estimates increases. This results in more effective guidance to the actor network to exert a more optimized control action. Training the critic target network, as shown briefly in Figure 3.14, aims to minimize the mean square error representing the estimated Q-value and sum of discounted cumulative rewards in order to mitigate overestimation bias and support stable learning for Q-value estimation.

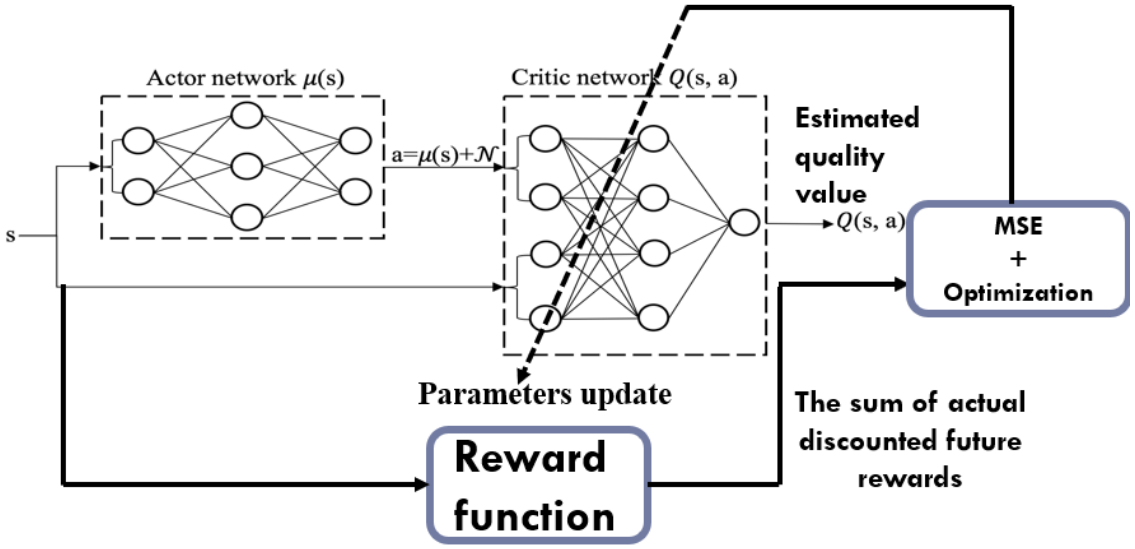


Figure 3.14 Overview on critic network optimization scheme

The proposed architecture, shown in Figure 3.15, is for both online and target critic networks. The architecture consists of three pathways [state path, action path, and main path], each handling distinct aspects of the network's function. The separate states and action pathways are useful in detecting different mappings because the observed states and output control action have varying natures.

The first pathway, “States path,” handles the state inputs by learning essential representations needed to estimate the Q-value function. The suggested architecture of this network is shown in Figure 3.15, where the input states are processed through a series of five fully connected layers with (125, 100, 75, 50, and 5 neurons, respectively) followed by the tanh activation function. The second pathway, “Action Path,” is responsible for processing the action input to learn the essential patterns needed for Q-value function estimation. The action path consists of four successive fully connected layers with (100, 75, 50, and 5 neurons, respectively) each followed by a tanh activation layer. The architecture ends with a single-neuron output layer representing the estimated / target Q-value. According to [104], the total number of tunable parameters for each critic network is 36,571, resulting in a shorter computational training time as compared to the usage of the topology suggested in [52] and [60].

Both pathways converge at the main path, where the states and action pathways' outputs are added. The combined data is then further processed through another fully connected layer of 5 neurons to increase the complexity of patterns extracted from both paths. A tanh activation function follows this layer. The network's final output estimates the Q-value, effectively representing the agent's expected future rewards based on the state-action pair.

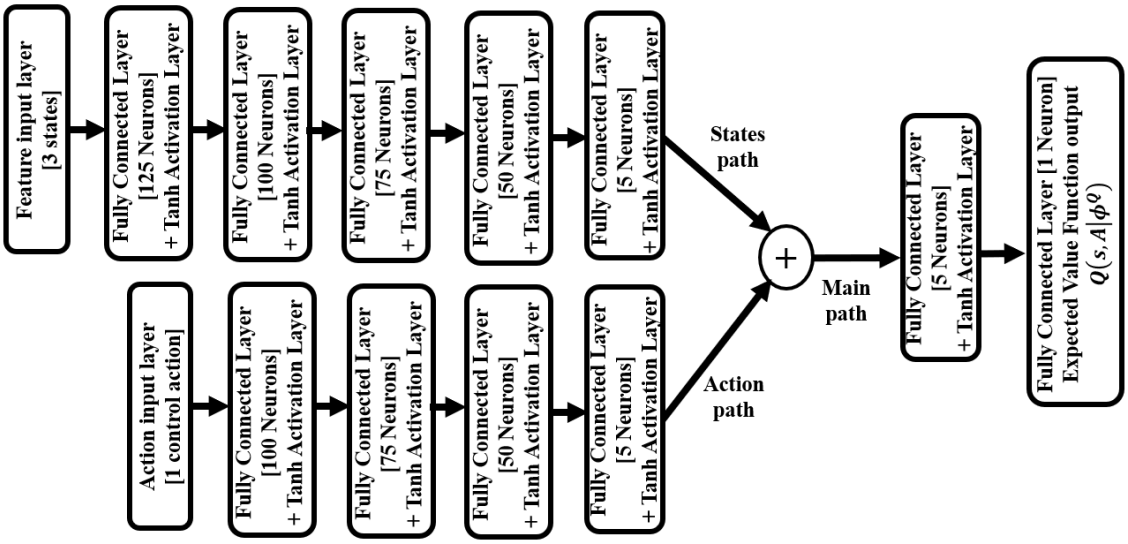


Figure 3.15 Neural network architecture of the critic network

3.4.5. Reward Function Formulation

The reward function is a scalar value that indicates how close the agent is in achieving the required goal based on the current states [53]. The objective of training is to maximize the cumulative sum of rewards. Choosing a reward function and its state representation is one of the most challenging parts as it depends on understanding the environment, states, and actions. The MSE, mean error (ME), only-positive (O-P), and positive-negative (P-N) in [33] have been tested on our continuous-action WT problem to see their behavior. Also, a suggested logic-based reward mechanism is tested to keep the generator speed and output power at rated levels as mentioned in (3.87). The logic-based reward function has been found to provide the finest results during training.

There are two main types of shaping reward functions, which are the dense and sparse functions. The dense reward functions give frequent, incremental feedback for actions, facilitating rapid learning. The sparse reward functions offer feedback only at significant milestones, making learning slower [53]. The logic-based reward function used is considered a dense reward function in our problem, as feedback is provided to the agent at every time step based on the current states.

The inputs to the reward function must be informative and relevant to the environment. The rotor speed and generator output power are chosen as inputs. The selected inputs differ from the states provided to the critic network but overlap in rotor speed as a state for both. In [33], The generator output power estimates the states. In [107], the generator power of each turbine in the wind farm is a function of control action and states. As a result, the generator output power, in our case, depends on the

pitch control action a_t and the states s_t. This implies that the reward function r_t is a function in both a_t and s_t too.

Our goal is to operate around the rated rotor speed and rated generator output power. This is done by minimizing error bands $\left||\Delta\omega_{rotor}|\right|$ and $\left||\Delta P_{Gen}|\right|$. Equation (3.87) describes the behavior of the actual reward function received. The error bands are not required to be large, thus increasing fluctuations around rated values, nor are they very small, leading to a sparse reward function. The target error bands for both rotor speed and generator output power are chosen by trial and error to be $\pm$ 0.3% of the rated values. The training process based on the logic-reward function helped the agent focus on moving the environment rotor speed and generator output power states to operate at the closest range to the rated values with minimum possible fluctuations.

3.4.6. Fuzzy Logic of F-DDPG

One of the inherent challenges with the model-free control strategy, as utilized by the DDPG algorithm for wind turbine pitch control, is the complexity of conducting systematic analysis. This limitation arises because such analysis traditionally relies on detailed mathematical modelling of the system being controlled [108]. To overcome such a challenge, a fuzzy system is implemented to offer a smooth transition between different trained agents to handle the perturbations of the wind inflow and uncertainties of WT dynamics to ensure the controller's robustness at all operating conditions. Due to a lack of general-purpose design methodologies, the design of fuzzy control systems is typically done using heuristic procedures [109]. The design is based on the measure of disparity between the wind turbine dynamics due to perturbations in the external wind inflow. Based on the gap-metric correlation matrix between the models stated in Table 2.3 of Subsection 2.2.4, six operating points at mean wind speeds of (14, 16, 18, 20, 22, and 24) m/s are chosen for training the DDPG agents. The optimal control action of F-DDPG is obtained based on a fuzzy system that processes the six-trained DDPG agents' control actions. The triangular membership function with a 0.5 cut is used [110]. Figure 3.16 illustrates the shape of the membership function, where the centers of the membership functions are the average wind speeds. The interference of control action regions between each of two consecutive wind speed breakpoints represents how the ratio of each agent control action is selected. Based on the instantaneous wind speed, the optimal control action is defuzzied according to the mean. The fuzzy control law C_f represents the input collective pitch control action to the wind turbine model, as shown in Figure 3.16 and equation (3.102).

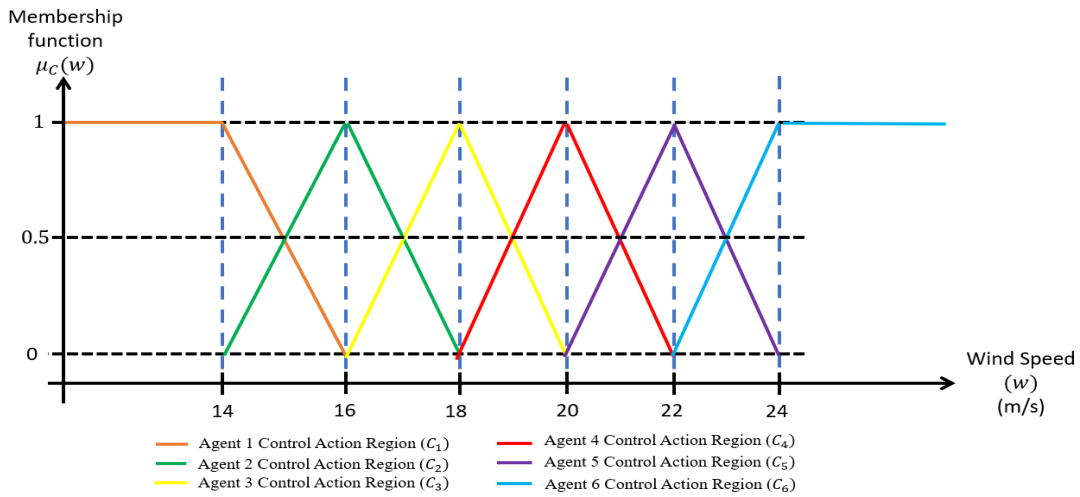


Figure 3.16 Triangular-membership fuzzy logic function of the optimal control action

$$C_f = \begin{cases} C_1 & , \ w < 14 \\ 0.5 * \left((16 - w) * C_1 \ + \ (w - 14) * C_2\right), & 14 \leq \ w < 16 \\ 0.5 * \left((18 - w) * C_2 \ + \ (w - 16) * C_3\right), & 16 \leq \ w < 18 \\ 0.5 * \left((20 - w) * C_3 \ + \ (w - 18) * C_4\right), & 18 \leq \ w < 20 \\ 0.5 * \left((22 - w) * C_4 \ + \ (w - 20) * C_5\right), & 20 \leq \ w < 22 \\ 0.5 * \left((24 - w) * C_5 \ + \ (w - 22) * C_6\right), & 22 \leq \ w < 24 \\ C_6 & , \ 24 \leq \ w \end{cases} \quad (3.102)$$

3.4.7. Proposed F-DDPG Controller training results

The training phase involves the development of six distinct DDPG agents within a medium-fidelity onshore environment. Each agent focuses on a specific average wind speed. The number of episodes in the training of each agent is 300. The episode duration is 100 seconds. The agent with the highest reward, as indicated in Table 3.8, is selected among the different episodes. A stochastic wind profile of 30000 seconds (8.33 hours) is generated for training. The wind medium-turbulence model used in the training phase is the National Wind Technology Center model [111].

The cumulative rewards gathered by DDPG agents during training at different wind speed profiles are evaluated against the total rewards earned when FMPC runs throughout the same profiles. The performance enhancement indicator in percentage, which refers to performance development, is calculated based on the formula (3.103):

$$\text{Performance enhancement indicator} = \frac{J_{F-DDPG} - J_{FMPC}}{J_{FMPC}} * \mathbf{100} \quad (3.103)$$

J_{F-DDPG} and J_{FMPC} are the cumulative rewards of F-DDPG and FMPC, respectively. F-DDPG has outperformed the FMPC in terms of cumulative rewards. Figure 3.17 displays a sample learning curve during the training of a DDPG agent across all episodes, specifically at an average wind speed of 22 m/s. Initially, the first five episodes utilized FMPC for control guidance, achieving an estimated cumulative reward of 35,000. Subsequently, the DDPG agent took over control actions completely

and reached an estimated cumulative reward of 52,000 by the end of the training episodes. The training of the agent quickly converged, with the cumulative reward reaching a consistent optimum level after roughly 25 episodes. Table 3.8 and Figure 3.18 summarize the training process of the six DDPG agents at various average wind speeds.

The training of the DDPG agents achieved different levels of cumulative reward functions at various wind speeds, as depicted in Figure 3.14. This variation and the gap-metric analysis suggest we are essentially managing six distinct WT systems. Consequently, the integration of a fuzzy system becomes crucial. It blends the control actions from all trained agents, ensuring the optimal control action is effectively applied across all operating conditions.

Table 3.8 Comparison between the cumulative reward of using FMPC and trained F-DDPG agents over 300 episodes.

Average Wind Speed (m/s)	Cumulative reward of using FMPC	Cumulative Reward of DDPG Agents during training	Percentage of performance Enhancement (%)	Max episode reward of DDPG Agents
14	62472	82148	31.49	97720
16	58319	69479	19.14	77680
18	53464	63995	19.70	74850
20	45683	56362	23.38	64760
22	37931	52571	38.60	62590
24	32612	43915	34.66	52340

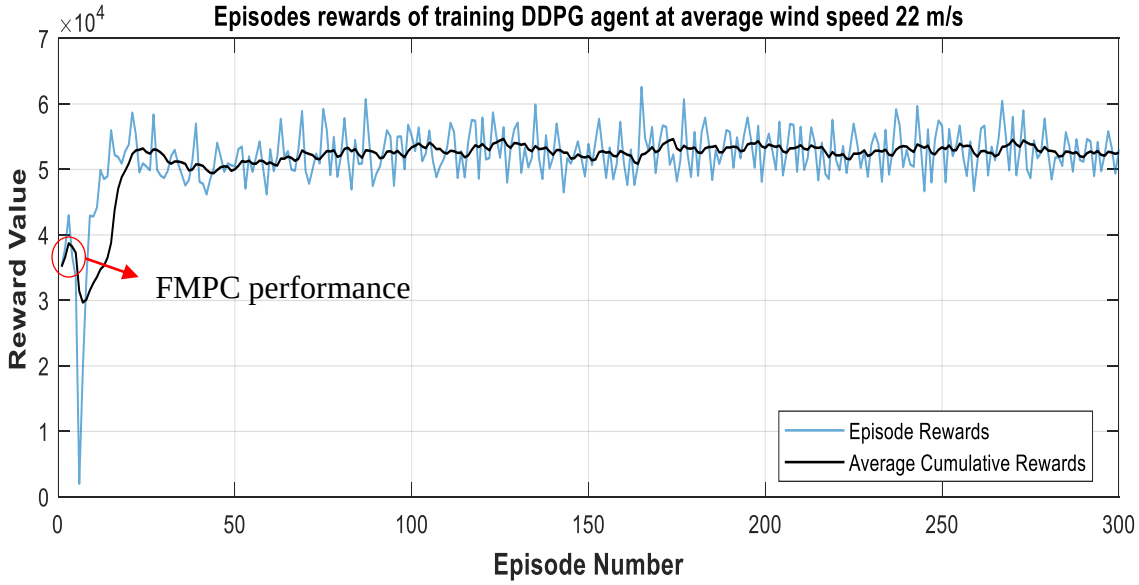


Figure 3.17 Episodes and average cumulative rewards of the DDPG agent training at an average wind speed 22 m/s.

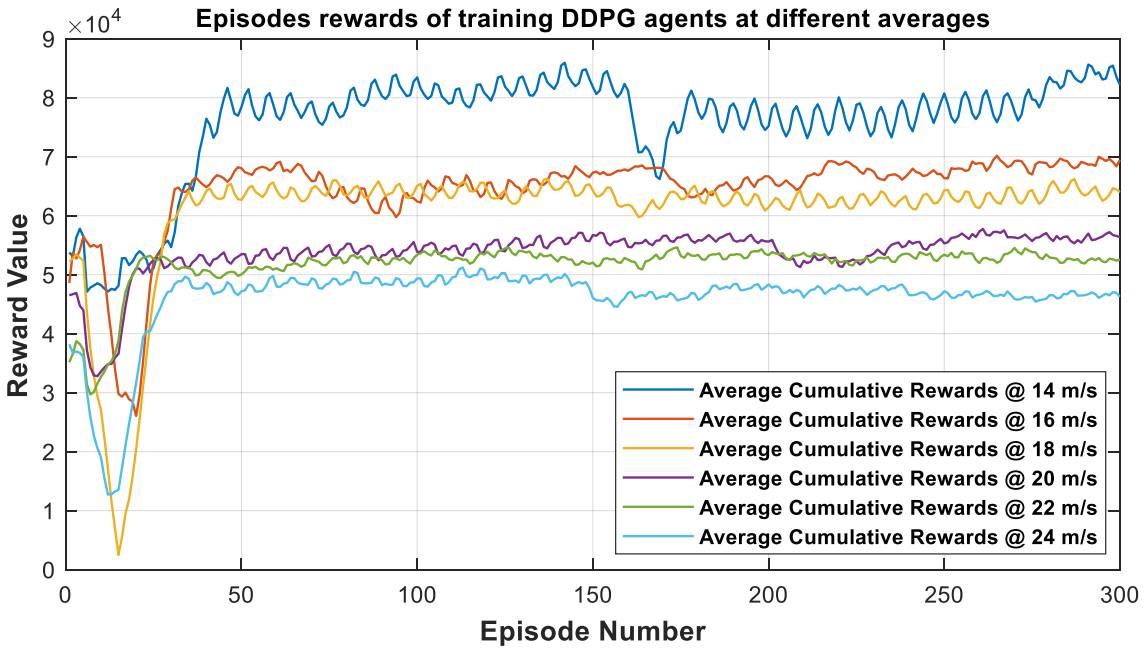


Figure 3.18 Cumulative episodes rewards reflecting the training progress of the DDPG agents at different averages wind speed profiles

Chapter 4 : Simulation Results

4.1. Introduction

The following results are based on high-fidelity 5-MW wind turbine models from the National Renewable Energy Lab's Open-FAST package, integrated with MATLAB/Simulink, specifically in region three operation mode [5]. Open-FAST, a tool certified by the National Renewable Energy Lab, accurately reflects real-world conditions. The "OC3-Hywind spar buoy" 5MW offshore and the 5MW onshore turbine models are utilized, representing advanced wind turbine simulation scenarios. The OC3-Hywind model, representing floating offshore turbines, is particularly noted for its complex dynamics behavior. In addition to complex hydrodynamics, the model combines high-turbulent wind and external wave conditions. The simulations encompass two scenarios: the first with an offshore turbine using the IEC Kaimal wind profile, designed for high-turbulent oceanic conditions, and the second with an onshore turbine employing the Great Plains Low-Level Jet (GP_LLJ) wind profile for moderate turbulence. Both scenarios' stochastic wind profiles are 3-dimensional models that are exported from running the Turbsim tool [43]. The wind profiles for both cases, as depicted in Figures 4.1 and 4.2, encompass stochastic wind speeds from 12 m/s to 25 m/s to demonstrate a generalized and realistic version of results for all potential wind speeds in region 3. The durations for both scenarios are 3000 seconds.

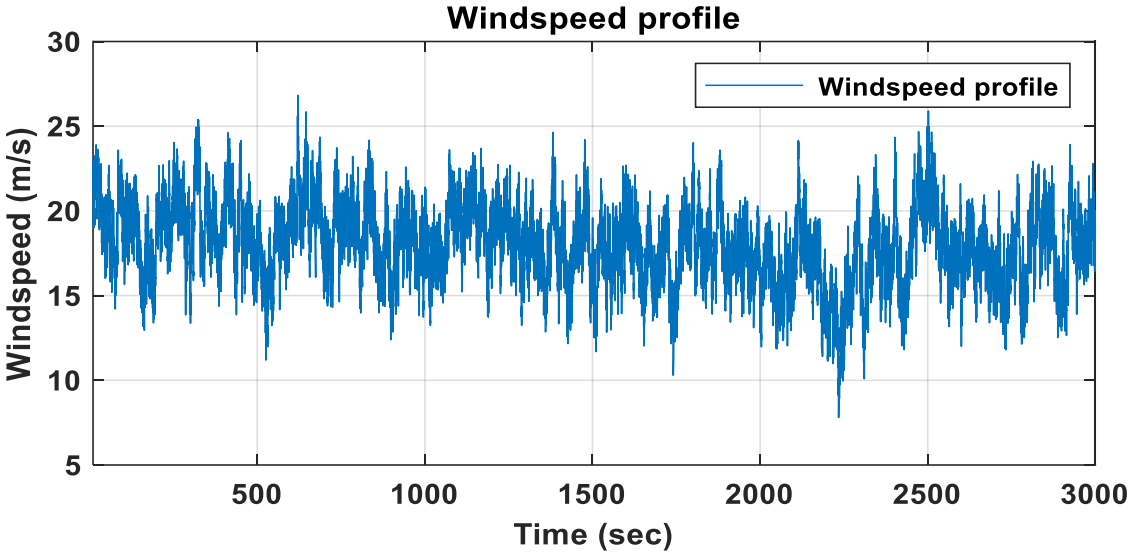


Figure 4.1 IEC Kaimal high-turbulent wind profile (Scenario I offshore)

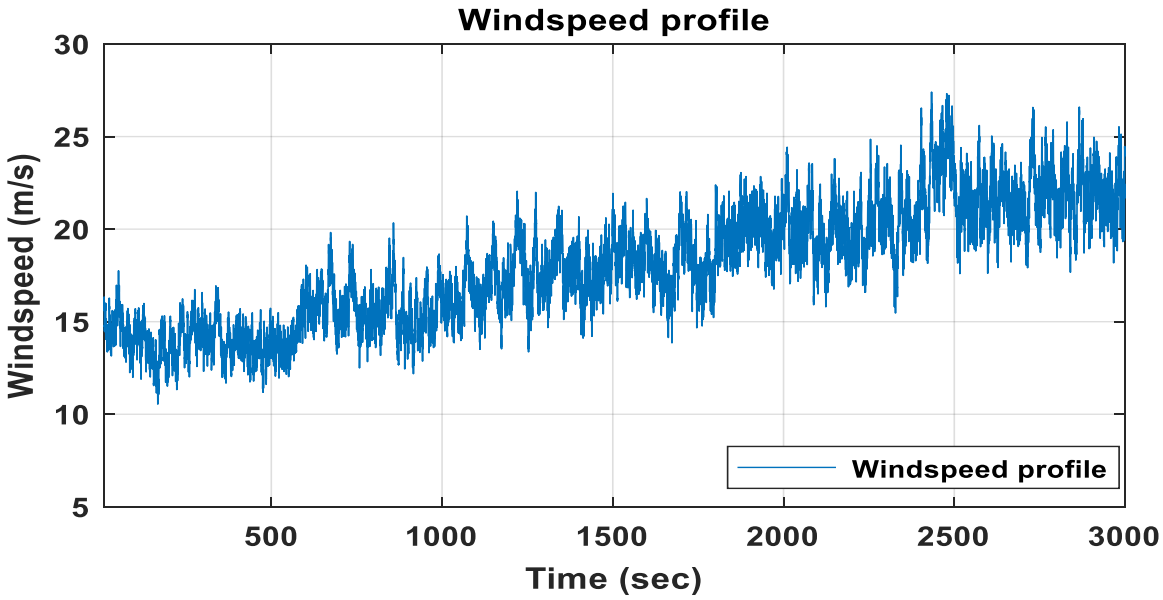


Figure 4.2 GP_LLJ moderate-turbulent (Scenario II onshore)

4.2. The proposed CNN-C simulation results

In this Subsection, the performance of the proposed CNN-C is compared with that of the GSPI controller and FMPC. Subsections 4.2.1 and 4.2.2 focus on the simulation results obtained in offshore and onshore scenarios.

4.2.1. Scenario I performance analysis (offshore test)

In this scenario, the simulations have been conducted on the high-fidelity Open-FAST floating offshore spar buoy WT model. The IEC Kaimal wind profile, chosen for its highly turbulent nature over ocean surfaces, spans all wind speed ranges in region three and is simulated for 3000 seconds, as depicted in Figure 4.1. The proposed CNN-C's performance is compared to the FMPC and GSPI controllers. The study focuses on six key aspects: generator speed, output power, collective pitch control signals, optimality index, simulation time, and economic feasibility. A typical numerical analysis, encompassing a comprehensive comparison between the controllers' performance, is summarized in Subsection 4.4.1, especially in Table 4.1.

Regarding the generator speed, the proposed CNN-C demonstrates notable proficiency in stabilizing the generator speed at its rated value, exhibiting fewer fluctuations compared to the FMPC and GSPI controllers. This is evident in Figure 4.3 and Table 4.1, where the achieved standard deviation (STD), peak-to-peak error, and maximum values are consistently lower for the proposed controller. Although all controllers function within comparable rpm ranges, the CNN-C uniquely sustains an average generator speed closest to the rated value, exhibiting a minimal deviation of 0.0006%. This contrasts the FMPC and the GSPI controller, which show a significantly higher average deviation of 0.010 % and 0.013, respectively. Notably, the proposed CNN-C displays significantly the slightest fluctuations with an STD value of 6.091. However, the FMPC and the GSPI controller exhibit STD of 9.647 and 17.760, respectively.

Concerning generator output power, the proposed CNN-C performs better in sustaining the generator power close to the rated value, exhibiting fewer fluctuations than alternative controllers. As illustrated in Figure 4.4, the CNN-C optimally refines power generation, maintaining fluctuations within the range of [4500-5000] kW along almost all wind profile periods. This stands in contrast to other controllers, which experience fluctuations in the range of [3500-5100] kW. Significantly, the proposed controller surpasses others by attaining the highest average power extraction, measured at 4890 kW, outperforming the FMPC's 4833 kW and the GSPI's 4709 kW. This is reflected in the energy harvesting from the WT system. Additionally, it demonstrates superior performance in terms of the lowest standard deviation, peak-to-peak error, and maximum values relative to other controllers, as emphasized in Table 4.1.

Concerning the collective pitch control signals, as depicted in Figure 4.5, it is evident that the collective pitch control signals from both the proposed CNN-C and other controllers follow a remarkably similar pattern. Notably, the STD is lowest for the GSPI controller.

The metric utilized to assess the performance of various controllers in terms of optimality is known as the optimality index. This index is derived from the reward function, which is designed to be maximized. Essentially, it represents the total reward accumulated by each controller over the entire simulation duration of 3000 seconds. In this context, the proposed controller has achieved the top score of 16.261 on this index, surpassing the FMPC's 12.456 and the GSPI's 5.704. This highlights its superior performance and higher optimality relative to the other controllers.

The total simulation time in MATLAB is composed of the initialization time, execution time and termination time. The simulation time is the factor that allows the assessment of the computational efficiency of each controller. Controllers that require less simulation time can be more desirable when processing power is limited. The simulation time depends mainly on the processing capability of the device on which the simulations are conducted. In terms of computational efficiency, given our available processing resources, the proposed CNN-C significantly outperforms others by achieving the lowest simulation time of 1734.820 seconds. This is compared to the FMPC and GSPI controllers, which recorded longer simulation times of 2560.809 seconds and 2091.618 seconds, respectively. Being evolved from the FMPC, the proposed CNN-C enhanced the computational time compared to the FMPC and the GSPI controllers by a factor of 1.476 and 1.205, respectively. This emphasizes a less time latency.

Regarding the economic aspects of WT operations, we refer to the European Wind Energy Association's data, which indicates an average capacity factor of 41% for offshore turbines [112]. Assuming a consistent wind profile as depicted in Figure 4.1 throughout the year, our analysis indicates that the proposed CNN-C generates the highest estimated average annual energy output of 17.565 GWh. This is in comparison to the FMPC's production of 17.360 GWh and the GSPI controller's output of 16.916 GWh. With the levelized energy cost set at 11 US cents per kWh [113], the anticipated annual revenue from energy sales for the CNN-C is estimated at 1.9322 million USD, versus 1.9096 million USD for the FMPC and 1.86076 million USD for the GSPI controller. This analysis ensures the economic viability and superiority of the proposed CNN-C in offshore wind turbine applications.

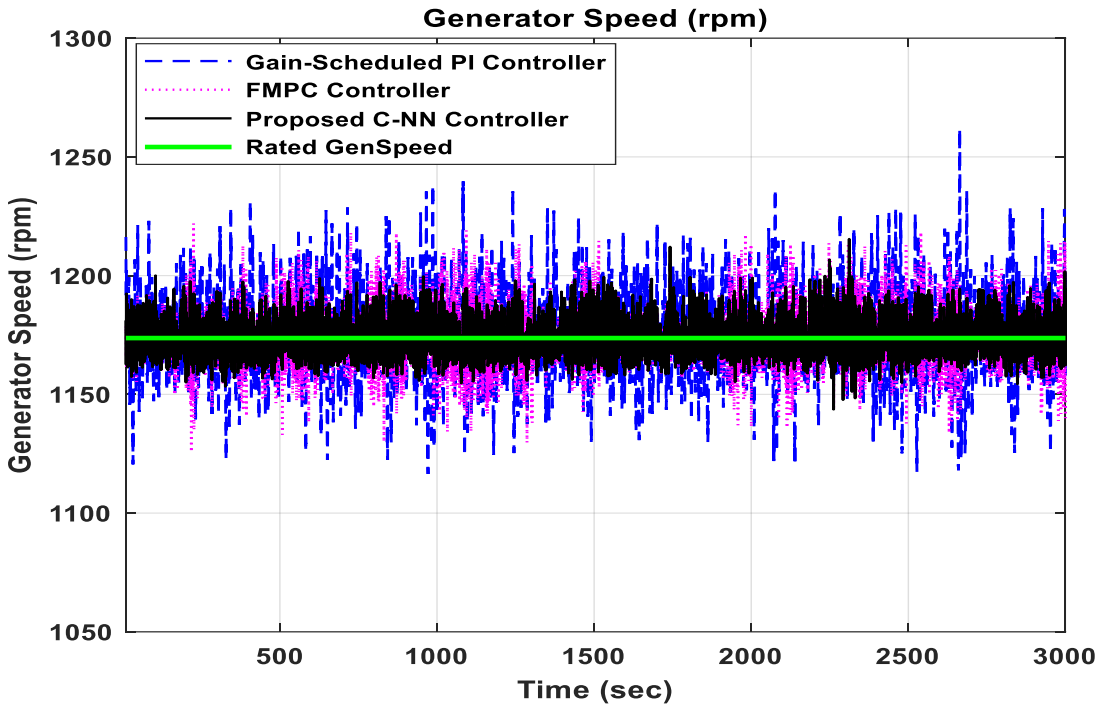


Figure 4.3 Comparison regarding generator speed. CNN-C in Scenario I (offshore).

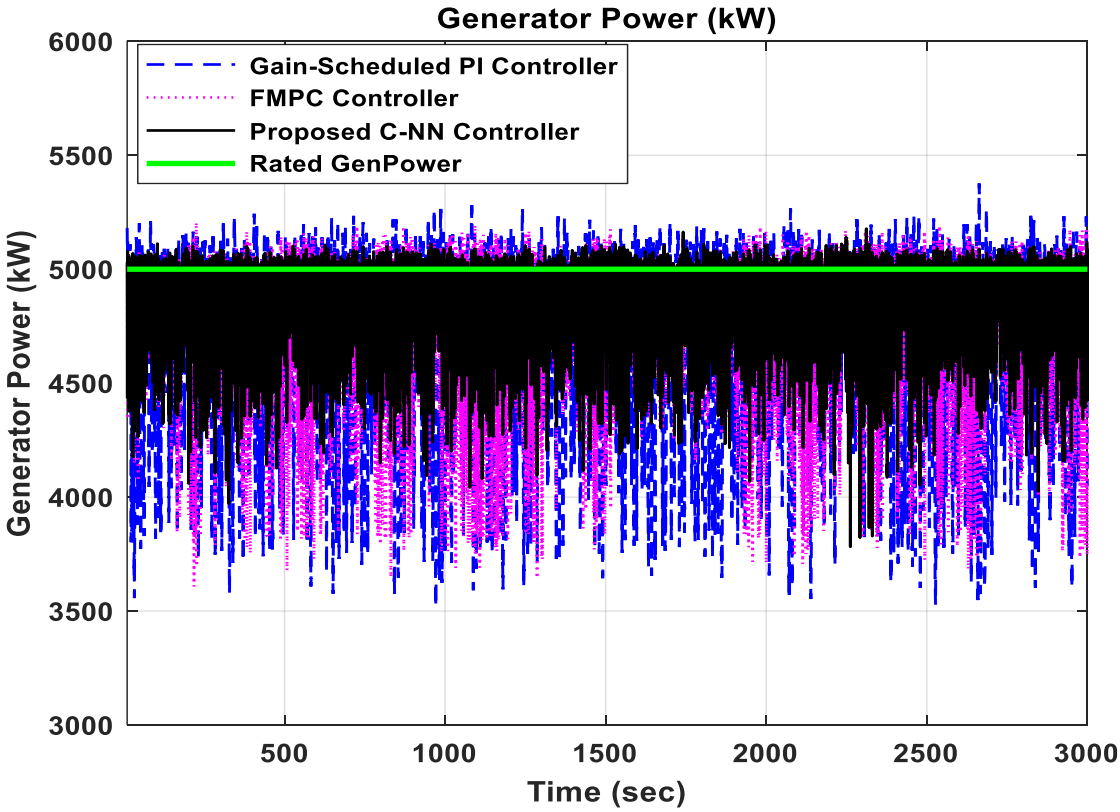


Figure 4.4 Comparison regarding generator output power. CNN-C in Scenario I (offshore).

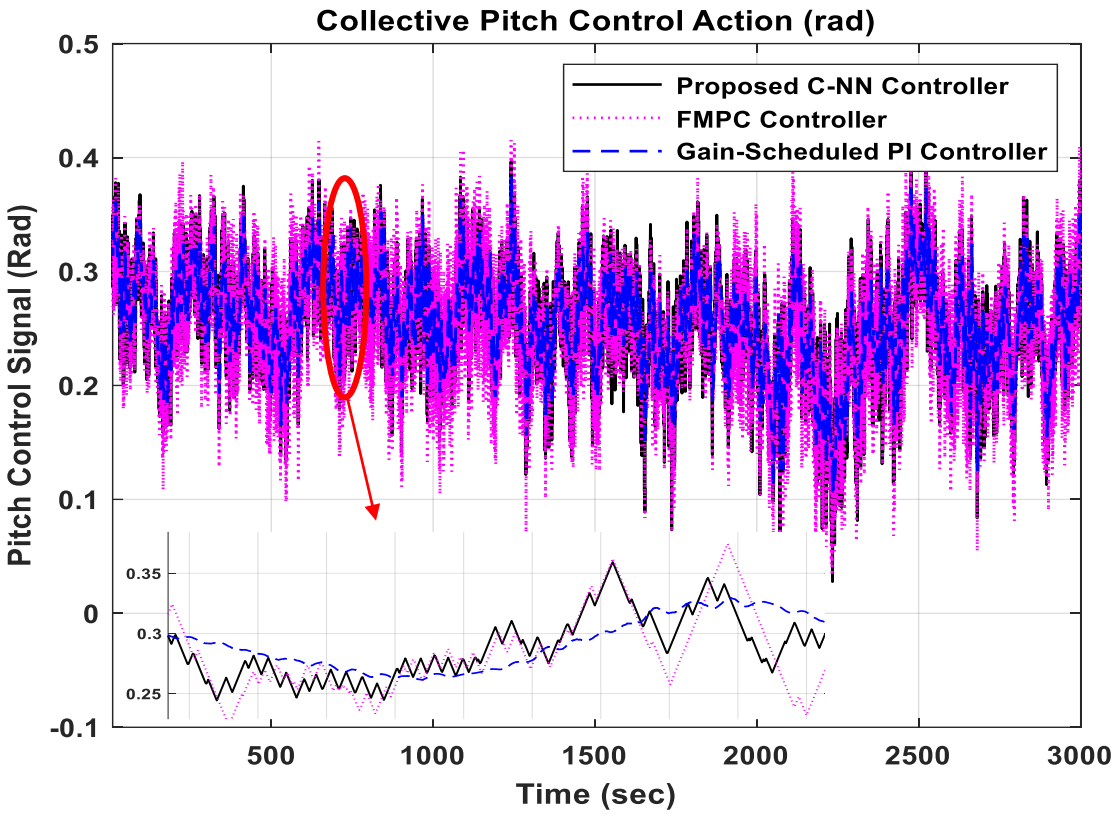


Figure 4.5 Comparison regarding collective pitch control signal. CNN-C in Scenario I (offshore).

4.2.2. Scenario II performance analysis (onshore test)

In this particular scenario, simulations have been conducted using the high-fidelity Open-FAST model of an onshore WT. This Subsection presents the outcomes of applying the CNN, FMPC, and GSPI controllers to this onshore WT model. The employed wind profile, known as GP_LLJ, is characterized by moderate turbulence. It is implemented at different stochastic averages of wind speed operating points to cover all possible wind speed ranges in region 3. The duration of the total wind profile is 3000 seconds, as depicted in Figure 4.2. We focus on the same aspects used in Scenario I to assess the proposed CNN-C and other controllers' performance.

Regarding the generator speed aspect, the proposed CNN-C performance is almost identical to the FMPC performance but excels in the GSPI controller performance. As depicted in Figure 4.6, the proposed controller and the FMPC demonstrate nearly identical patterns with fewer fluctuations than the GSPI controller. While all controllers operate within similar rpm ranges, the FMPC and CNN-Cs maintain an average generator speed nearest the designated rated value, with only a slight deviation of 0.058% and 0.06%, respectively. This contrasts with the GSPI, which shows a slightly higher average deviation of 0.062%. Notably, the GSPI controller displays significantly the highest fluctuations with an STD value of 18.920, while both the FMPC and the proposed CNN-C exhibit STD values of 4.321 and 4.351, respectively.

Concerning generator output power, the FMPC and the proposed CNN-C exhibit superior and identical performance in sustaining the generator power close to the rated

value, exhibiting fewer fluctuations than the GSPI controller. As illustrated in Figure 4.7, the FMPC and the CNN-C optimally refine power generation, maintaining fluctuations within the range of [4500-5000] kW along almost all wind profile periods. This contrasts the GSPI controller, which experiences fluctuations in the range of [4000 - 5100] kW. Additionally, Figure 4.7 clearly shows that fluctuations intensify as wind speed escalates. Remarkably, both the proposed controller and the FMPC outperform the GSPI controller by achieving the highest and nearly identical average power extraction values, with the proposed controller at 4943.291 kW and the FMPC at 4942.871 kW, significantly higher than the GSPI's 4709.963 kW. This is reflected in the energy harvesting from the WT system.

Concerning the collective pitch control signals, as depicted in Figure 4.8, it is evident that the collective pitch control signals from both the proposed CNN-C and other controllers follow a remarkably similar pattern. Notably, the STD value is lowest for the GSPI controller.

Regarding optimality, the FMPC reached the highest rating with a score of 24.25, closely followed by the proposed CNN-C, which scored 24.16 on this metric, significantly outperforming the GSPI's score of 6.945. All scores are multiplied by $x10^5$. This highlights the proposed CNN-C's superior performance and a higher level of optimality relative to the GSPI controller.

In terms of computational efficiency, given our available processing resources, the proposed CNN-C significantly outperforms others by achieving the lowest simulation time of 741.143 seconds. This is compared to the FMPC and GSPI controllers, which recorded longer simulation times of 1614.447 seconds and 864.556 seconds, respectively. The proposed CNN-C enhanced the computational time compared to the FMPC and the GSPI controllers by a factor of 2.178 and 1.166, emphasizing less latency.

Regarding the economic aspects of WT operations, we refer to the European Wind Energy Association's data, which indicates an average capacity factor of 24% for onshore turbines [112]. Under the assumption of the consistent wind profile shown in Figure 4.2 throughout the year, our analysis shows that the proposed CNN-C slightly outperforms the FMPC controller, achieving the highest estimated average annual energy production of 10.395 GWh, marginally higher than the FMPC's 10.394 GWh, and significantly surpassing the GSPI controller's output of 9.905 GWh. Setting the levelized energy cost at 7 US cents per kWh [113], the anticipated annual revenue from energy sales for the proposed CNN-C is estimated at 1.1435 million USD, versus 1.1433 million USD for the FMPC and 1.08955 million USD for the GSPI controller. This analysis ensures the economic viability and superiority of the proposed CNN-C in onshore wind turbine applications.

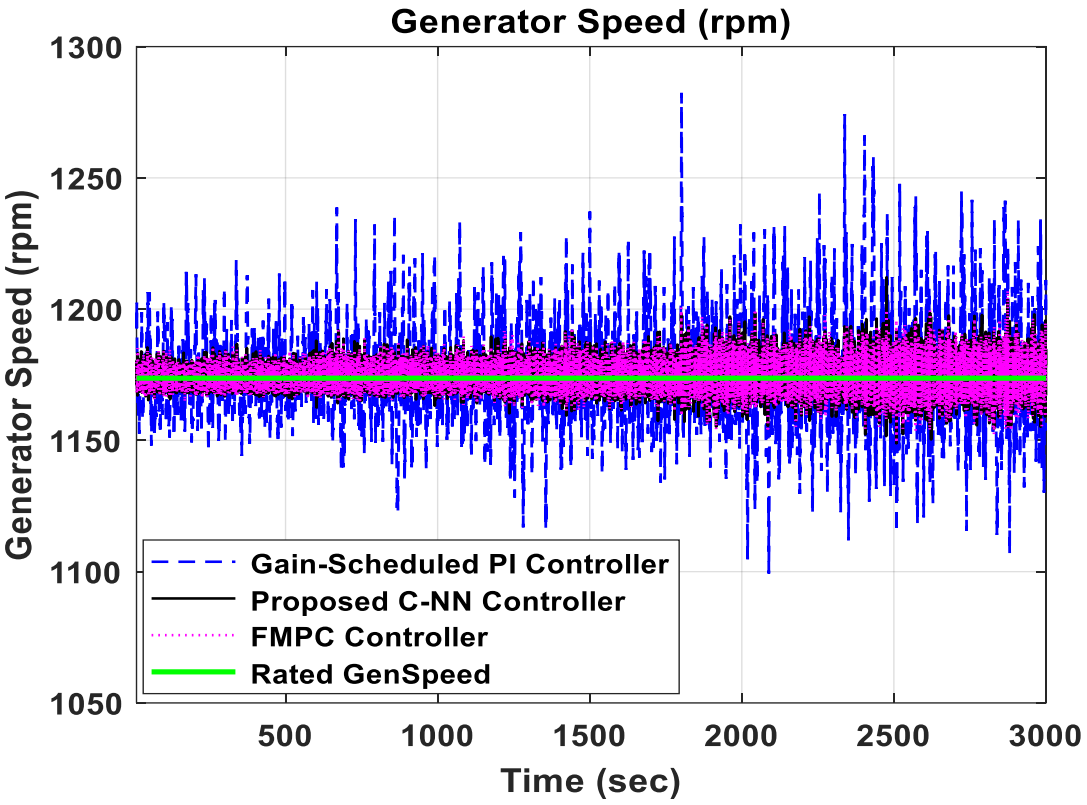


Figure 4.6 Comparison concerning generator speed. CNN-C in Scenario II (onshore).

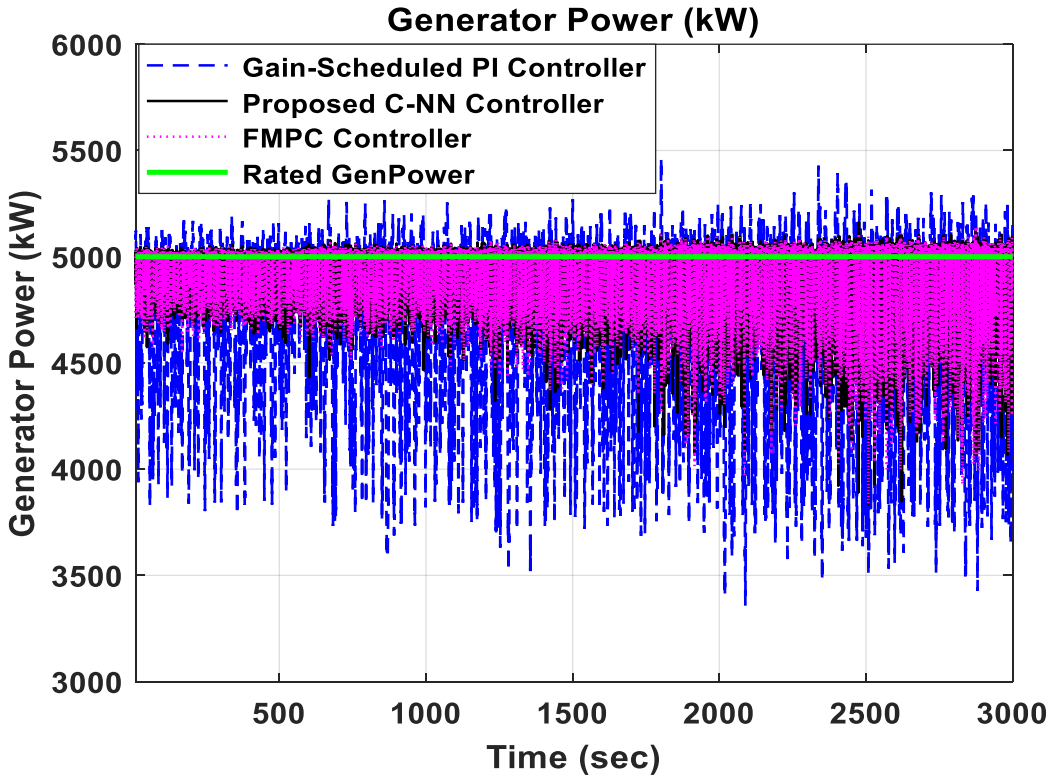


Figure 4.7 Comparison regarding generator output power. CNN-C in Scenario II (onshore).

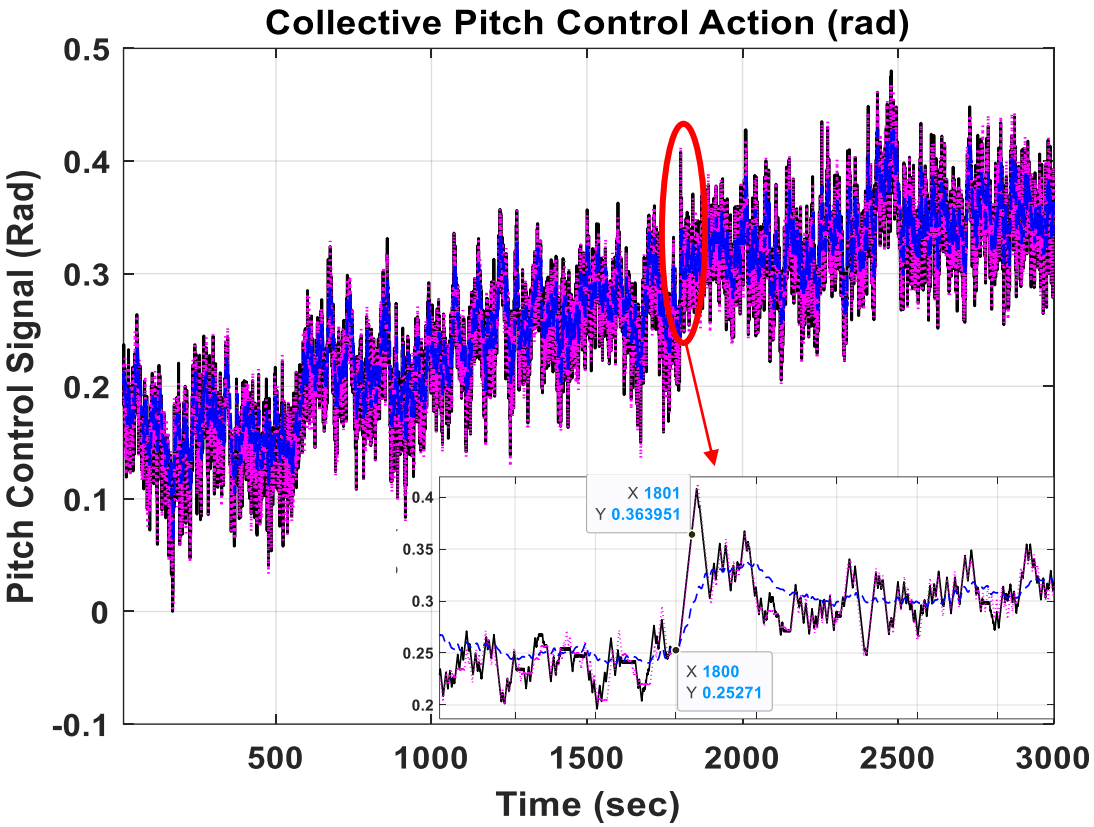


Figure 4.8 Comparison regarding collective pitch control signal. CNN-C in Scenario II (onshore).

4.3. The proposed F-DDPG simulation results

This section summarizes the proposed F-DDPG Controller performance results. Subsections 4.3.1 and 4.3.2 are dedicated to performance analysis for the first and second testing case scenarios. They compare the proposed F-DDPG controller's performance alongside other controllers, including GSPI [114], LQR [115], and a single-DDPG-agent (DDPG controller) trained at an average wind speed of 18 m/s [19].

4.3.1. Scenario I performance analysis (offshore test)

In this scenario, the simulations have been conducted on the same conditions and aspects mentioned in Subsection 4.2.1. A typical numerical analysis for the offshore test, encompassing a comprehensive comparison of the controllers' performance, is summarized Subsection 4.4.1, especially in Table 4.1.

Regarding the generator speed, the proposed F-DDPG controller demonstrates notable proficiency in stabilizing the generator speed at its rated value, exhibiting fewer fluctuations compared to the GSPI, LQR, and DDPG controllers. This is evident in Figure 4.9 and is summarized in Table 4.1, where the achieved standard deviation (STD), peak-to-peak error, and maximum values are consistently lower for the proposed controller. Despite all controllers operating within similar rpm ranges, the F-DDPG controller maintains a mean generator speed closer to the rated value, with only a minor deviation of 0.081% less than the LQR and DDPG controllers but slightly

higher than the GSPI. Notably, the DDPG controller displays the highest mean value deviation among the controllers, as indicated in Table 4.1, while the LQR achieves the highest STD compared to the other controllers.

Concerning generator output power, the proposed F-DDPG controller exhibits superior performance in sustaining the generator power at its rated value, exhibiting fewer fluctuations than alternative controllers. As illustrated in Figure 4.10, the F-DDPG controller optimally refines power generation, maintaining fluctuations within the range of [4500-5000] kW along almost all wind profile periods. This stands in contrast to other controllers, which experience fluctuations in the range of [3500-5100] kW. Notably, the proposed controller achieves the highest mean extracted power value, accompanied by the lowest STD, peak-to-peak error, and maximum values when compared to alternative controllers.

Concerning the collective pitch control signals, as depicted in Figure 4.11 and further stated in Table 4.1, it is evident that the collective pitch control signals from both the proposed F-DDPG controller and other controllers follow a remarkably similar pattern. Notably, the STD is highest for the DDPG controller. However, it should be highlighted that the F-DDPG controller exhibits marginally more fluctuations than the GSPI controller and LQR.

The metric utilized to assess the performance of various controllers in terms of optimality is known as the optimality index. This index is derived from the reward function, which is designed to be maximized. Essentially, it represents the total reward accumulated by each controller over the entire simulation duration of 3000 seconds. As indicated in Table 4.1, the proposed controller has attained the highest value on this index, signifying its superior performance and greater optimality compared to other controllers.

Given our available processing resources, the proposed F-DDPG controller exhibits a limitation in terms of computational efficiency. The simulation time recorded the longest interval by 3694.906 seconds. This is compared to the GSPI, LQR, and DDPG controllers, which recorded less simulation times of 2091.618, 2359.363 seconds, and 2118.820 seconds, respectively. Thus, the proposed F-DDPG controller has the highest complexity in execution time.

Regarding the economic aspects of WT operations, we refer to the European Wind Energy Association's data, which indicates an average capacity factor of 41% for offshore turbines [112]. Under the assumption of the consistent wind profile shown in Figure 4.1 throughout the year, our analysis reveals that the F-DDPG controller yields the highest estimated average annual energy production of 17.764 GWh compared to other controllers. With the levelized energy cost set at 11 US cents per kWh [113], the expected annual energy sales are calculated as shown in Table 4.1. This analysis ensures the economic viability and superiority of the proposed F-DDPG controller in offshore wind turbine applications.

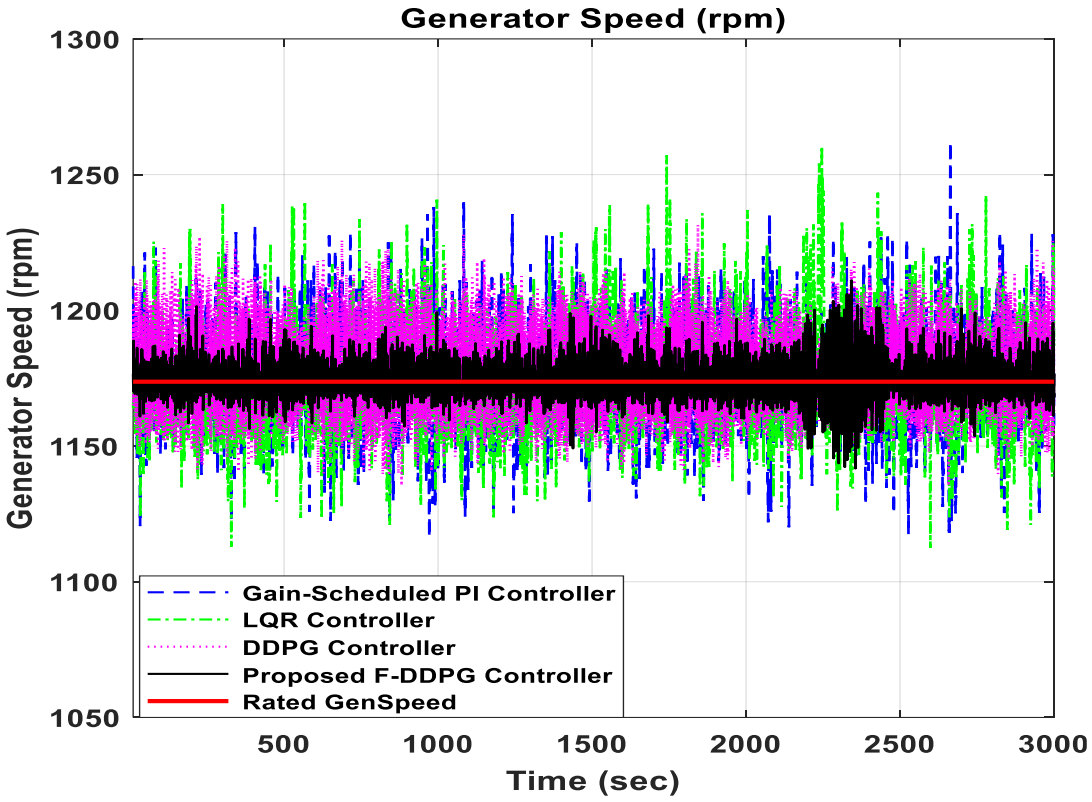


Figure 4.9 Comparison concerning generator speed. F-DDPG in Scenario I (offshore model)

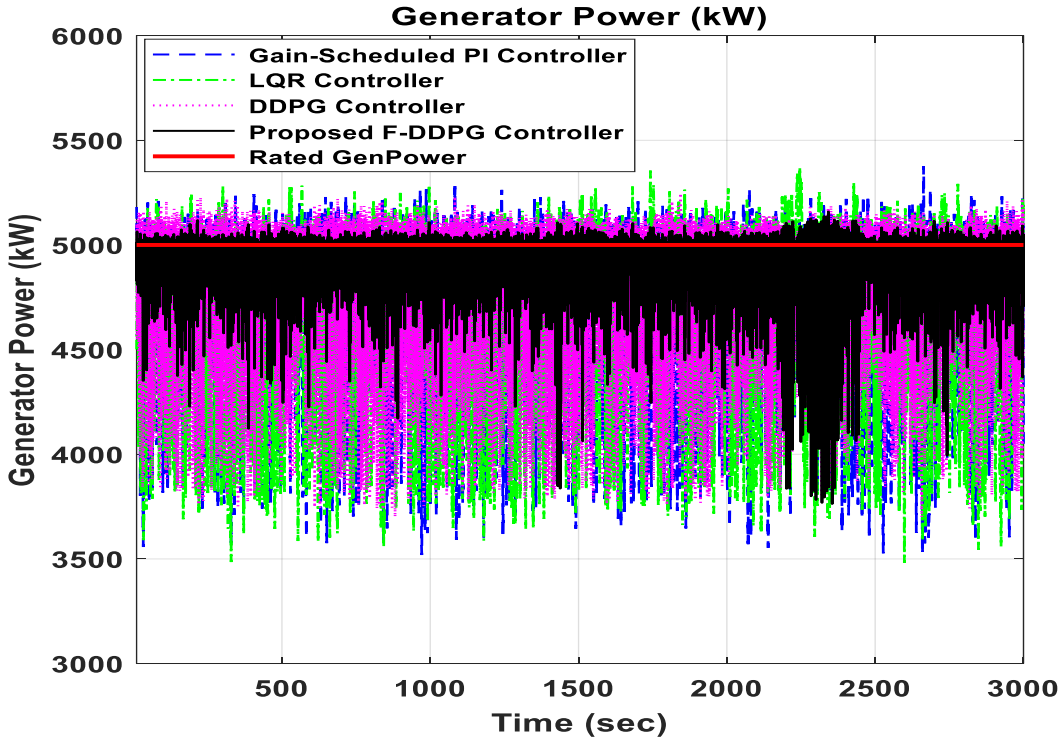


Figure 4.10 Comparison concerning generator power. F-DDPG in Scenario I (offshore model)

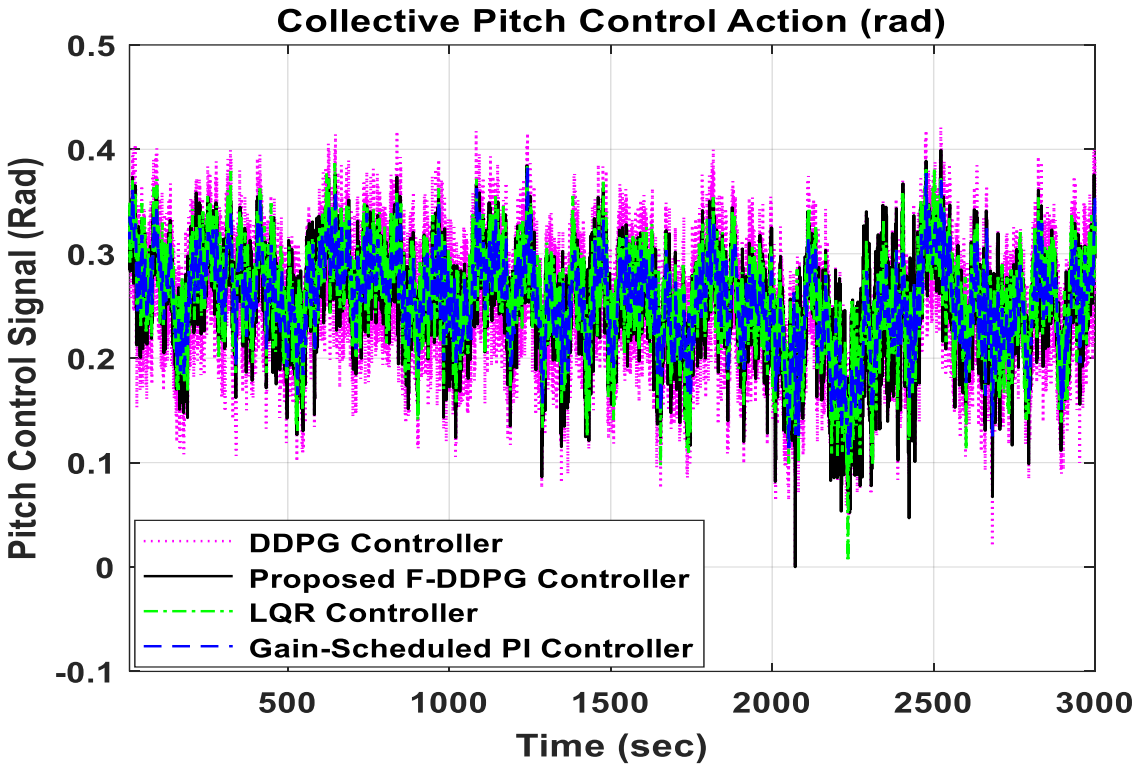


Figure 4.11 Comparison concerning the pitch angle signals. F-DDPG in Scenario I (offshore model).

4.3.2. Scenario II performance analysis (onshore test)

In this particular scenario, simulations have been conducted on the same conditions and aspects mentioned in Subsection 4.2.1. A typical numerical analysis for onshore test, which encompass a comprehensive comparison between the controllers' performance, is summarized in Subsection 4.4.1, especially in Table 4.2.

Regarding the generator speed, the F-DDPG and LQR controllers demonstrate a notable reduction in fluctuations, effectively sustaining the generator speed at its rated values, as illustrated in Figure 4.12. Conversely, the GSPI controller exhibits the highest standard deviation (STD) and the most significant fluctuations among the controllers, with the DDPG controller following closely behind, as detailed in Table 4.2. Furthermore, the DDPG controller shows the most significant mean value deviation from the rated value compared to its counterparts, signaling a less optimal performance in this specific aspect.

Regarding generator power, both the DDPG and F-DDPG controllers excel, achieving the highest average power extraction, as indicated in Table 4.2. Meanwhile, the F-DDPG and LQR controllers exhibit the lowest fluctuations in generator power, as demonstrated in Figure 4.13. On the other hand, the GSPI controller shows the most significant fluctuations, as evidenced by its high STD value, and it also records the lowest power extraction. This combination of factors suggests that the GSPI controller's performance is less effective compared to the other controllers in this aspect.

Regarding the collective pitch control signals, as illustrated in Figure 4.14 and further detailed in Table 4.2, it becomes clear that the signals from the proposed F-DDPG controller and those from alternative controllers display a similar pattern.

In terms of optimality, as shown in Table 4.2, the proposed controller has attained the highest value on this index of 24.515, signifying its superior performance and greater optimality compared to GSPI, LQR and DDPG controllers which have scored 5.704, 5.507 and 7.078, respectively.

In terms of computational efficiency, given our available processing resources, the proposed F-DDPG controller exhibits a limitation in this aspect. The simulation time recorded the longest interval by 3694.906 seconds. This is in comparison to the GSPI, LQR, and DDPG controllers, which recorded less simulation times of 2091.618, 2359.363 seconds and 2118.820 seconds, respectively. Thus, the proposed F-DDPG controller has the highest complexity in execution time.

Regarding the economic aspects of WT operations, we refer to the European Wind Energy Association's data, which indicates an average capacity factor of 24% for onshore turbines [112]. Under the assumption of the consistent wind profile shown in Figure 4.2 throughout the year, our analysis reveals that the proposed F-DDPG and DDPG controllers yield the highest estimated average annual energy production of 10.421 and 10.476 GWh compared to other controllers. The expected annual energy sales are calculated with a levelized cost of 7 US cents/kWh [113], as shown in Table 4.2.

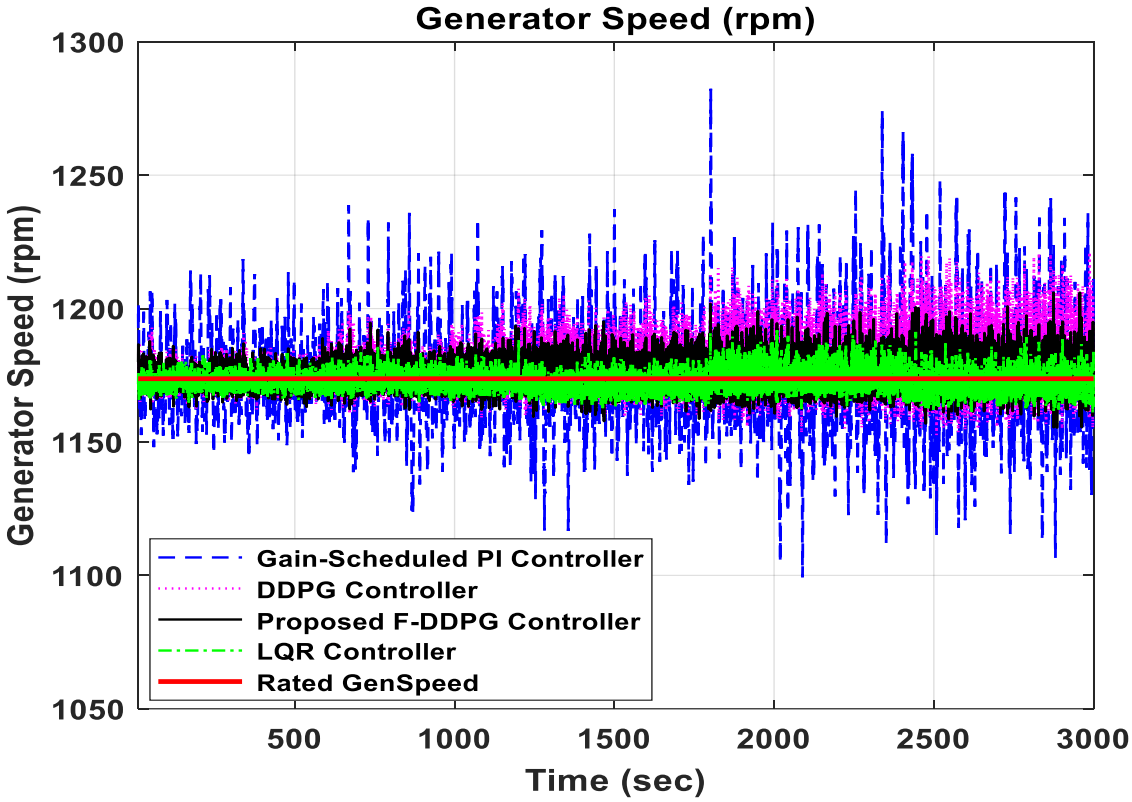


Figure 4.12 Comparison concerning generator speed. F-DDPG in Scenario II (onshore).

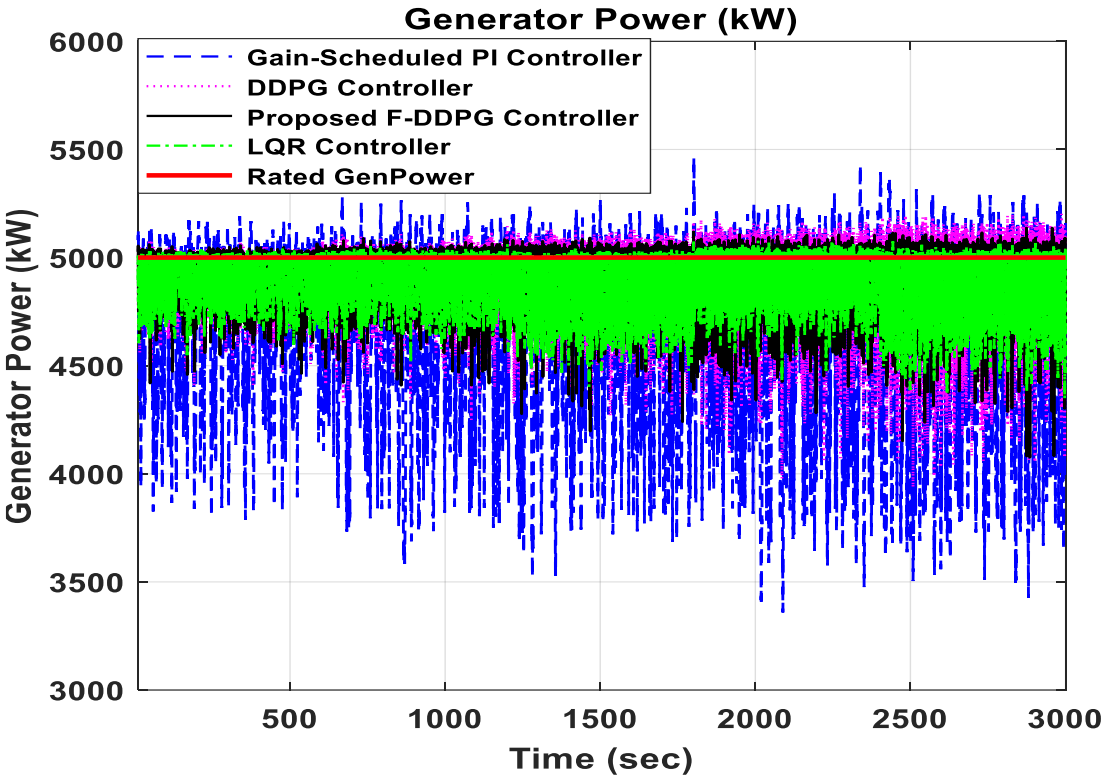


Figure 4.13 Comparison concerning generator power. F-DDPG in Scenario II (onshore).

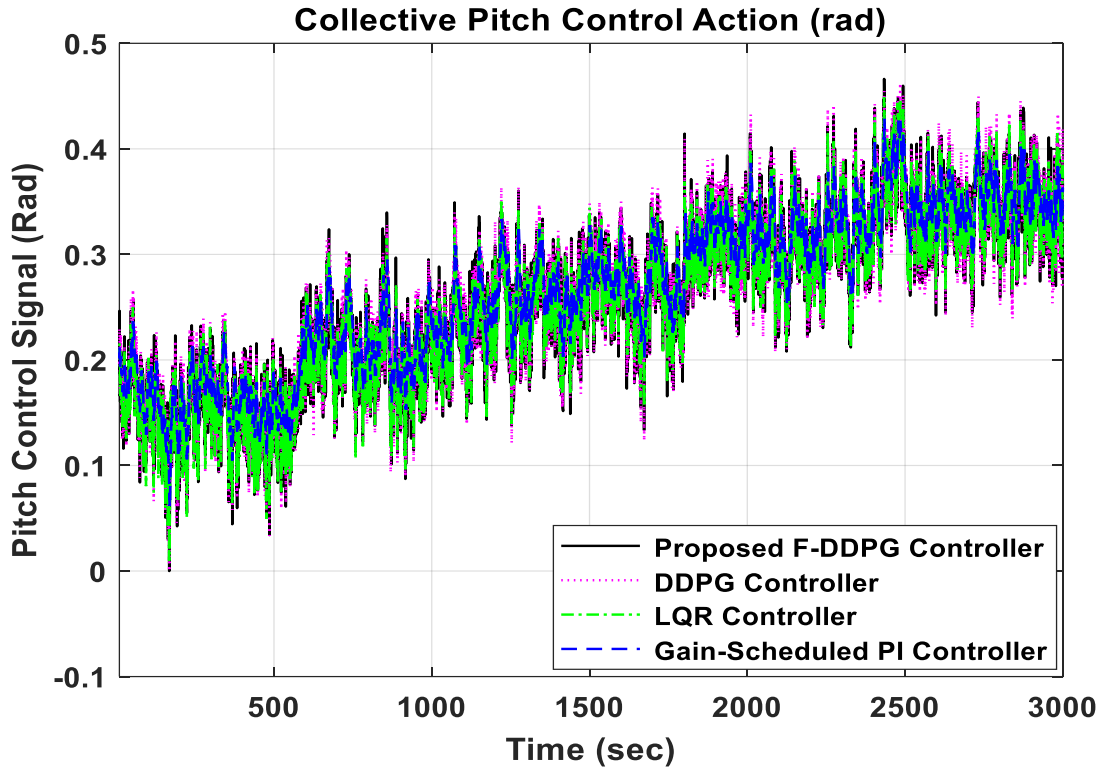


Figure 4.14 Comparison concerning the pitch angle control signals. F-DDPG in Scenario II (onshore).

4.4. Comprehensive numerical performance analysis

This section is intended to show a comprehensive numerical performance analysis of the proposed CNN and F-DDPG controllers compared to the other baseline controllers (GSPI, LQR, DDPG, and FMPC). Subsection 4.4.1 emphasizes the analysis of the results in the offshore and onshore test scenarios with numerical values. The intended enhancement in the performance of the two proposed controllers is presented in Subsection 4.4.2. The numerical results of different validation tests conducted in offshore and onshore cases are represented in Subsection 4.4.3.

4.4.1. The numerical results

The numerical results of the offshore and onshore model tests are summarized in Table 4.1 and Table 4.2, respectively. These tables address the generator speed, generator power, pitch angle command, optimality index, simulation time, total energy output and the expected energy sales are the main aspects. The values of the proposed methods are bolded in these tables.

Table 4.1 Statistical analysis of the offshore scenario test results.

Controllers Types		Scenario I (offshore model test)					
		GSPI	LQR	DDPG	FMPC	**CNN**	**F-DDPG**
Generator speed (rpm)	Mean	1173.855	1172.65	1178.446	1173.581	**1173.692**	**1174.649**
	|Mean value deviation| %	0.013	0.089	0.404	0.010	**0.0006**	**0.081**
	STD	17.760	17.92	12.902	9.647	**6.091**	**4.920**
	Peak-to-Peak	145.496	147.838	96.176	96.104	**71.760**	**69.288**
	Min	1116.471	1112.543	1135.313	1125.777	**1143.655**	**1141.723**
	Max	1261.967	1260.381	1231.489	1221.882	**1215.415**	**1211.011**
Electric power (kW)	Mean	4709.243	4669.984	4876.033	4833.149	**4890.294**	**4945.377**
	|Mean value deviation| %	5.815	6.600	2.479	3.337	**2.194**	**1.092**
	STD	433.242	432.279	295.920	270.335	**166.181**	**131.103**
	Peak-to-Peak	1857.317	1887.57	1546.307	1597.829	**1395.675**	**1396.048**
	Min	3518.702	3481.697	3699.878	3607.425	**3782.031**	**3762.898**
	Max	5376.019	5369.267	5246.186	5205.255	**5177.706**	**5158.946**
Pitch angle command (rad)	Mean	0.254	0.258	0.252	0.250	**0.254**	**0.253**
	STD	0.042	0.045	0.057	0.0532	**0.046**	**0.047**
	Peak-to-Peak	0.276	0.379	0.421	0.384	**0.373**	**0.399**
	Min	0.108	0.008	0.000	0.033	**0.028**	**0.000**
	Max	0.384	0.387	0.421	0.417	**0.400**	**0.399**
Optimality index cumulative reward ($x10^5$)		5.704	5.507	7.078	12.4557	**16.261**	**24.515**
Total elapsed simulation time (seconds)		2091.618	2359.363	2118.820	2560.809	**1734.820**	**3694.906**
Expected annual energy production (GWh)		16.916	16.775	17.514	17.360	**17.565**	**17.764**
Expected Energy sales in US dollars ($x10^6$)		1.86076	1.84525	1.92654	1.9096	**1.9322**	**1.95404**

Table 4.2 Statistical analysis of the onshore scenario test results.

Controllers Types		Scenario I (onshore model test)					
		GSPI	LQR	DDPG	FMPC	**CNN**	**F-DDPG**
Generator speed (rpm)	Mean	1174.428	1172.786	1178.879	1174.378	**1174.405**	**1175.103**
	|Mean value deviation| %	0.062	0.078	0.441	0.058	**0.060**	**0.119**
	STD	18.920	3.166	8.011	4.321	**4.351**	**4.605**
	Peak-to-Peak	183.086	31.029	68.558	58.036	**63.277**	**50.625**
	Min	1099.287	1160.828	1152.562	1149.051	**1148.875**	**1155.361**
	Max	1282.373	1191.858	1221.120	1207.088	**1212.153**	**1205.986**
Electric power (kW)	Mean	4709.963	4913.639	4981.652	4942.871	**4943.291**	**4955.265**
	|Mean value deviation| %	5.801	1.727	0.367	1.143	**1.134**	**0.895**
	STD	428.580	103.013	117.228	116.712	**116.664**	**102.693**
	Peak-to-Peak	2104.231	728.718	1264.756	1306.411	**1329.753**	**1061.629**
	Min	3358.719	4348.637	3937.255	3835.822	**3834.061**	**4075.909**
	Max	5462.950	5077.355	5202.011	5142.233	**5163.813**	**5137.539**
Pitch angle command (rad)	Mean	0.267	0.257	0.259	0.259	**0.259**	**0.257**
	STD	0.073	0.076	0.078	0.079	**0.079**	**0.077**
	Peak-to-Peak	0.367	0.439	0.462	0.469	**0.480**	**0.466**
	Min	0.062	0.001	0.000	0.000	**0.000**	**0.000**
	Max	0.428	0.449	0.462	0.469	**0.480**	**0.466**
Optimality index cumulative reward ($x10^5$)		6.945	21.883	15.128	24.25	**24.160**	**22.771**
Total elapsed simulation time (seconds)		864.556	737.344	1162.405	1614.447	**741.143**	**2551.107**
Expected annual energy production (GWh)		9.905	10.330	10.476	10.394	**10.395**	**10.421**
Expected Energy sales in US dollar ($x10^6$)		1.08955	1.1363	1.15236	1.1433	**1.1435**	**1.14631**

4.4.2. Performance Enhancement

The proposed methods have encountered significant enhancement in performance compared to the baseline controllers. The enhancement ratios of each proposed method are listed in Tables 4.3 and 4.4.

Table 4.3 Performance enhancement percentages of the proposed CNN-C over baseline controllers for both scenarios.

Enhancement Ratio (%) = $\frac{\lvert Controller_{value} - C\text{-}NN_{value}\rvert}{Controller_{value}} * 100$		Scenario I (offshore test)		Scenario II (onshore test)	
		GSPI	FMPC	GSPI	FMPC
Generator speed (rpm)	Mean value deviation	95.385	94.000	3.226	-
	STD	65.704	36.861	77.003	-
	Peak-to-Peak	50.679	25.331	65.439	-
	Min	2.435	1.588	4.511	-
	Max	3.689	0.529	5.476	-
Electric power (kW)	Mean extracted power	3.845	1.182	4.954	0.008
	Mean value deviation	62.270	34.252	80.452	0.787
	STD	61.642	38.528	72.779	-
	Peak-to-Peak	24.855	12.652	36.806	-
	Min	7.484	4.840	14.152	-
	Max	3.689	0.529	5.476	-
Optimality index enhancement %		185.081	30.551	247.876	-
Computational cost improvement %		17.058	32.255	14.275	54.093
Annual energy production improvement %		3.837	1.181	4.947	0.010
Energy sales improvement in US dollar		71440	22600	53950.000	200.000

Table 4.4 Performance enhancement percentages of the proposed F-DDPG controller over baseline controllers for both scenarios.

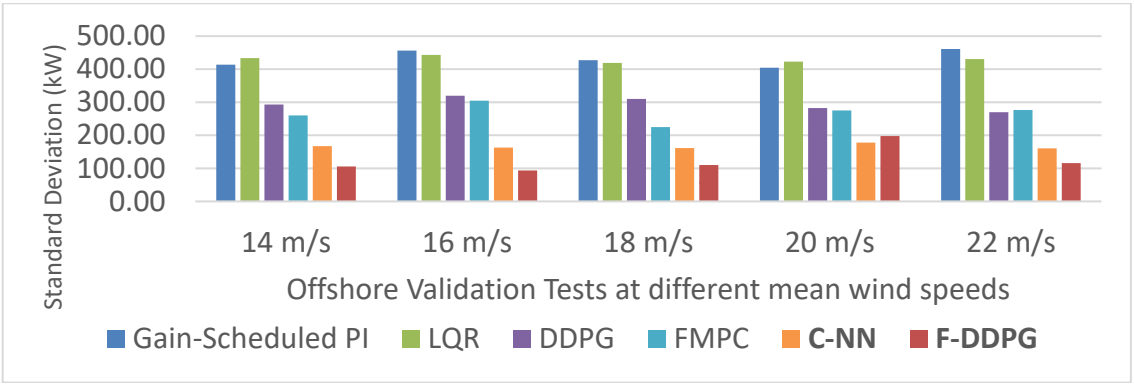
Enhancement Ratio (%) = $\frac{\lvert Controller_{value} - F\text{-}DDPG_{value}\rvert}{Controller_{value}} *$ 100		Scenario I (offshore model test)			Scenario II (onshore model test)		
		GSPI	LQR	DDPG	GSPI	LQR	DDPG
Generator speed (rpm)	Mean value deviation	-	8.989	79.951	-	-	73.016
	STD	72.297	72.545	61.866	75.661	-	42.517
	Peak-to-Peak	52.378	53.132	27.957	72.349	-	26.157
	Min	2.262	2.623	0.565	5.101	-	0.243
	Max	4.038	3.917	1.663	5.957	-	1.239
Electric power (kW)	Mean extracted power	5.014	5.897	1.422	5.208	0.847	-
	Mean value deviation	81.221	83.455	55.949	84.572	48.176	-
	STD	69.739	69.672	55.696	76.038	0.311	12.399
	Peak-to-Peak	24.835	26.040	9.717	49.548	-	16.061
	Min	6.940	8.077	1.703	21.353	-	3.522
	Max	4.038	3.917	1.663	5.957	-	1.240
Optimality index enhancement %		329.786	345.161	246.355	227.876	4.058	50.522
Annual energy production improvement %		5.013	5.896	1.427	5.209	0.881	-
Energy sales improvement in US dollar		93280	108790	27500	56760	10010	-

4.4.3. Validation Tests of controllers

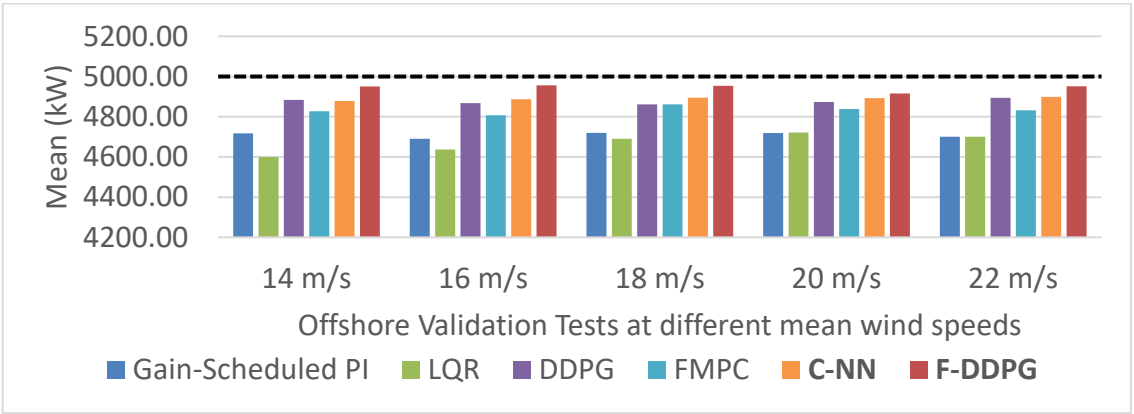
In order to ensure the controllers robustness and guarantee its stability, validation tests have been held. The proposed controllers' performance are compared with other baseline controllers. The tests are conducted on offshore and onshore WT models.

a) Offshore validation tests

Validation tests have been conducted using five distinct wind profiles, each lasting 600 seconds, at varying average wind speeds. The proposed and alternative controllers are evaluated across these wind profiles to affirm the effectiveness of both proposed controllers under diverse wind conditions. Regarding generator power, as illustrated in Figures (4.15-a) and (4.15-b), the proposed CNN-C and F-DDPG controllers consistently demonstrate the lowest standard deviation (STD) and highest average extracted power across all tested conditions. It is noticed that the CNN-C achieves higher power extraction and lower STD than its equivalent FMPC. In comparison, the F-DDPG demonstrates superiority in maximizing the power with the lowest STD among all controllers. Regarding generator speed, the proposed controllers achieve the lowest STD, as depicted in Figure (4.16-a), while Figure (4.16-b) indicates their mean speed closely aligns with the rated value. Conversely, the DDPG controller exhibits the most significant deviation in mean value from the rated speed, which is undesirable for WT system operation.

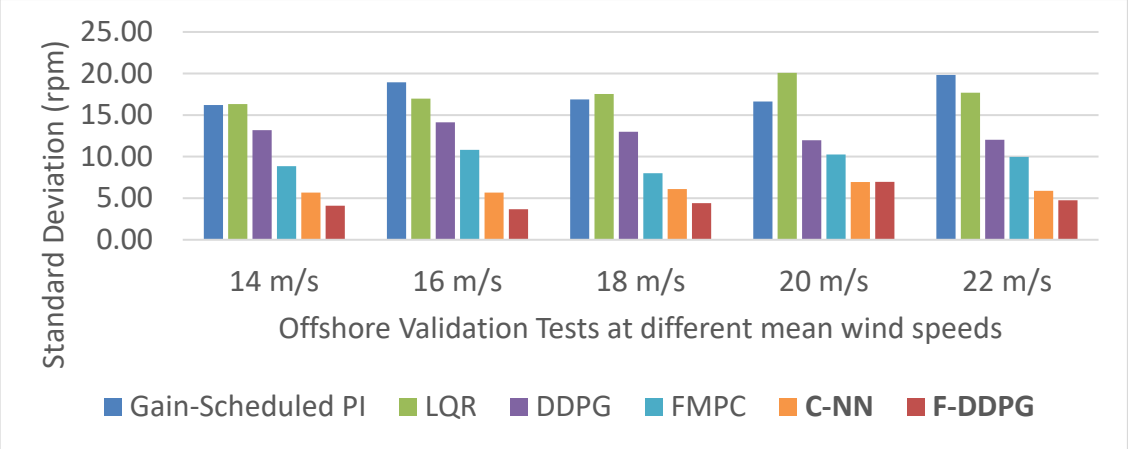


(a)

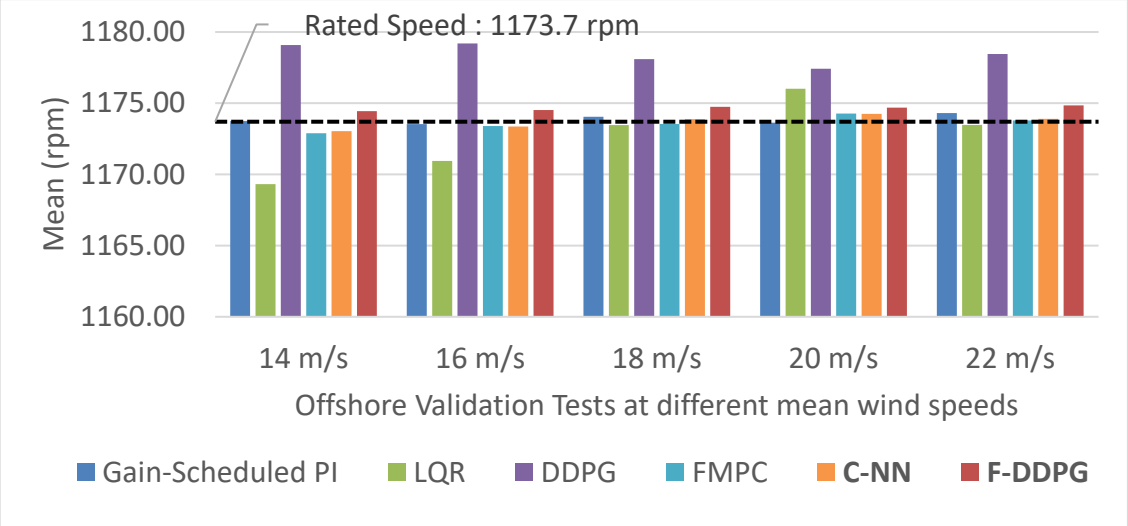


(b)

Figure 4.15 Offshore validation tests concerning generator power (a) STD value (b) Mean value



(a)



(b)

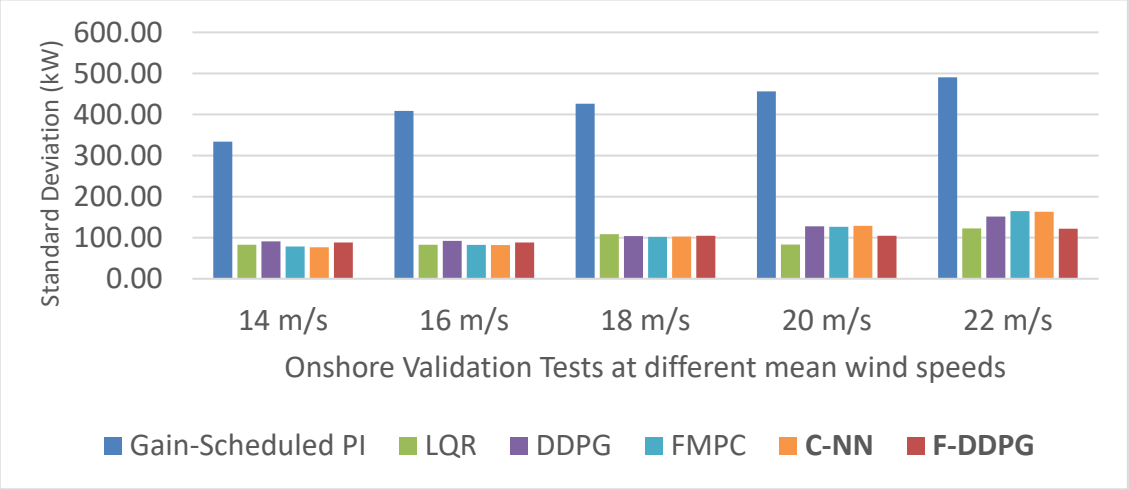
Figure 4.16 Offshore validation tests concerning generator speed. (a) STD value (b) Mean value

b) Onshore validation tests

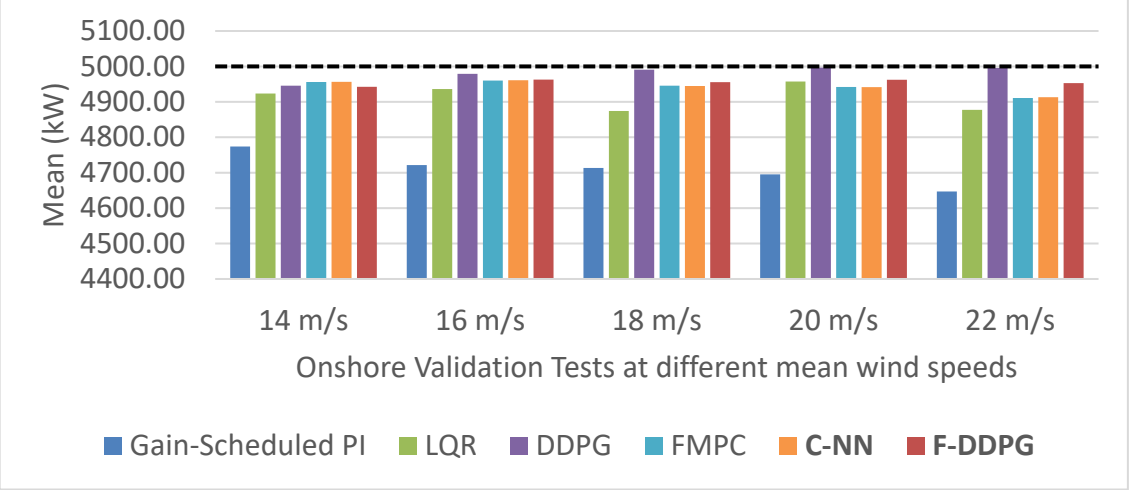
In these validation tests, the same aspects in Subsection (4.4.3-a) are considered. These tests were designed to evaluate the performance of the proposed controllers and their alternatives under a range of wind conditions, thereby verifying the effectiveness of the proposed controller. In terms of generator power, as illustrated in Figure (4.17-a), the F-DDPG exhibits notably low standard deviations compared to others in all operating points, indicating minimal fluctuations. In comparison, the CNN-C has almost an identical performance to the FMPC. In contrast, the GSPI controller shows the highest level of variability.

Furthermore, as shown in Fig. (4.17-b), both the F-DDPG and DDPG controllers achieve the highest mean power generation in all operating points, with the DDPG controller slightly outperforming. Concerning generator speed, depicted in Figure (4.18-a), the F-DDPG, CNN-C, FMPC, and LQR controllers are observed to have the lowest standard deviation values, indicating superior stability. Additionally, as presented in Figure (4.18-b), the CNN-C, FMPC and GSPI controllers are noteworthy

for having the slightest mean value deviations from the rated speed, setting them apart from the other controllers.

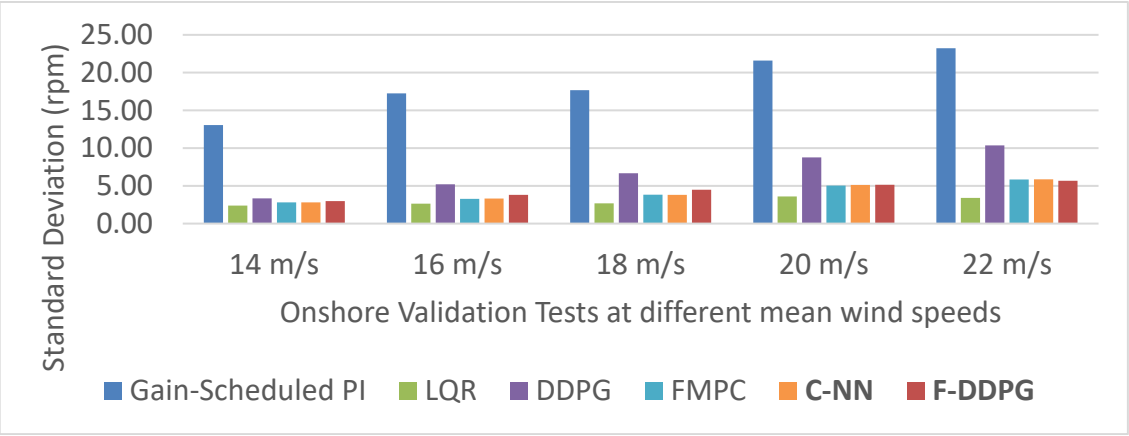


(a)

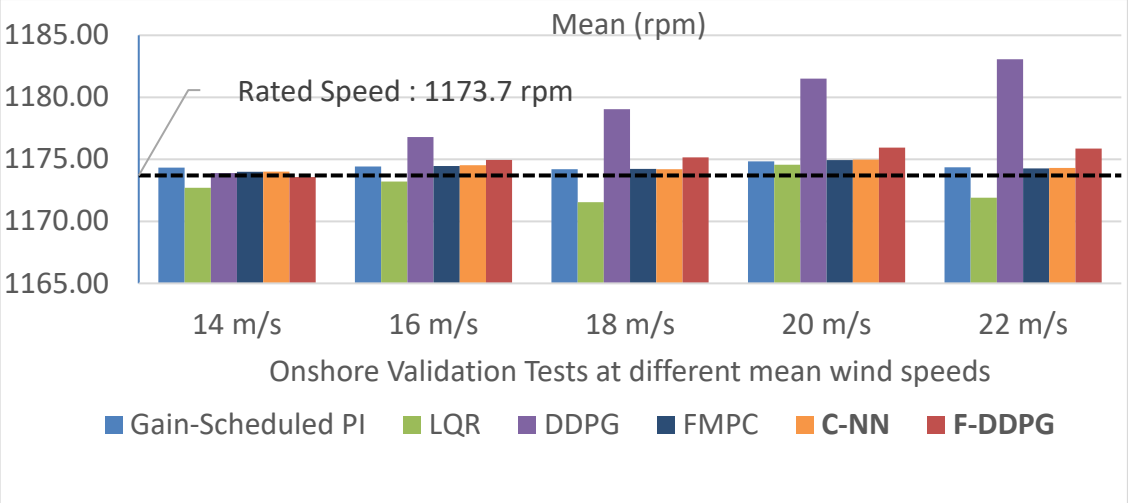


(b)

Figure 4.17 Onshore validation tests concerning generator power (a) STD value (b) Mean value



(a)



(b)

Figure 4.18 Onshore validation tests concerning generator speed. (a) STD value (b) Mean value

4.4.4. Ranking controllers according to different operational aspects

All controllers are compared and ranked according to different operational aspects. The aspects of ranking are considering mean value deviation of generator speed, generator speed and power fluctuations, computational efficiency, maximizing generator output power and the optimality of each controller. Here are definitions for each aspect shown in the spider chart, focusing on their relevance to collective pitch angle control in wind turbine systems:

1. Minimum Mean Value Deviation (Generator Speed):

    - This aspect measures how well the controller maintains the generator speed close to its reference value. Lower deviations indicate better performance in stabilizing the generator speed, which is crucial for efficient energy conversion and grid stability.

2. Generator Speed Fluctuations Handling:

    - This refers to the controller's ability to manage variations in generator speed due to changing wind conditions. Effective handling minimizes wear and tear on the generator and ensures a smoother power output.

3. Maximizing Generator Output Power:

    - This measures the controller's efficiency in maximizing the amount of power generated from the available wind energy. It involves optimizing the pitch angle to capture the maximum kinetic energy from the wind.

4. Generator Power Fluctuations Handling:

- This aspect evaluates how well the controller manages the fluctuations in power output. Effective control ensures a more stable power supply to the grid, which is important for maintaining grid reliability.

5. Dealing with Complex Hydrodynamics:

   - This measures the controller's capability to handle the complex fluid dynamics involved in wind energy conversion. It includes the ability to adapt to varying wind speeds, directions, and turbulence, which can significantly impact turbine performance.

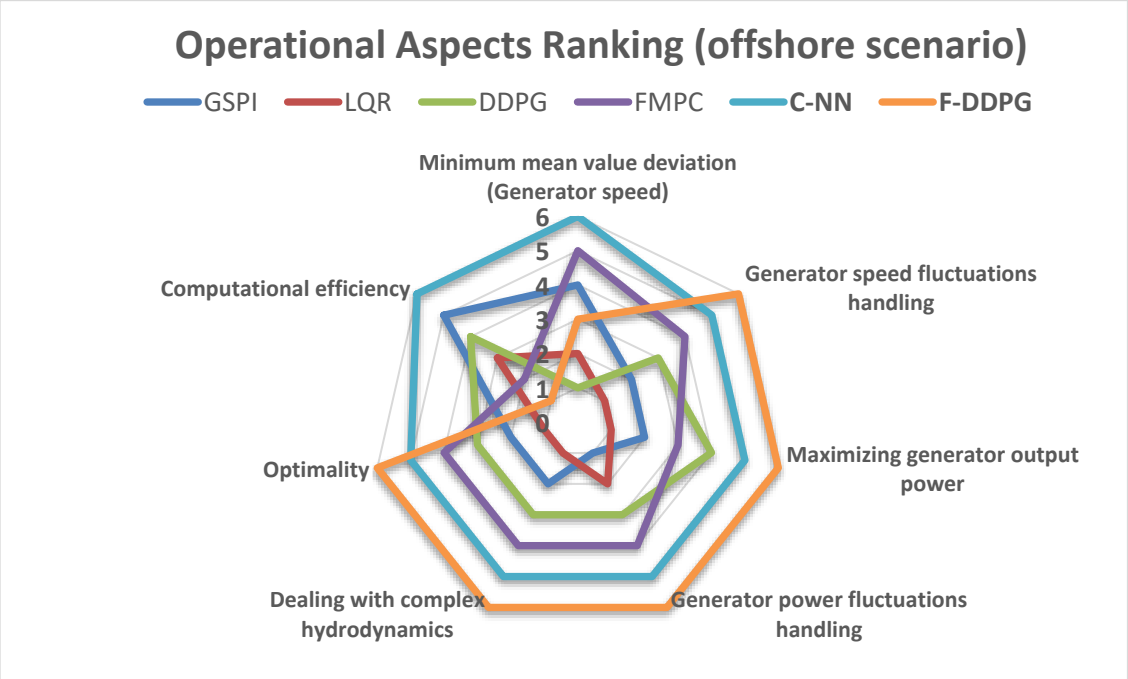
6. Optimality:

   - This refers to how well the controller optimizes the overall performance of the wind turbine system. It involves balancing various factors such as power output, rotor fluctuations, and operational stability to achieve the best possible performance. It's calculated based on the reward function in

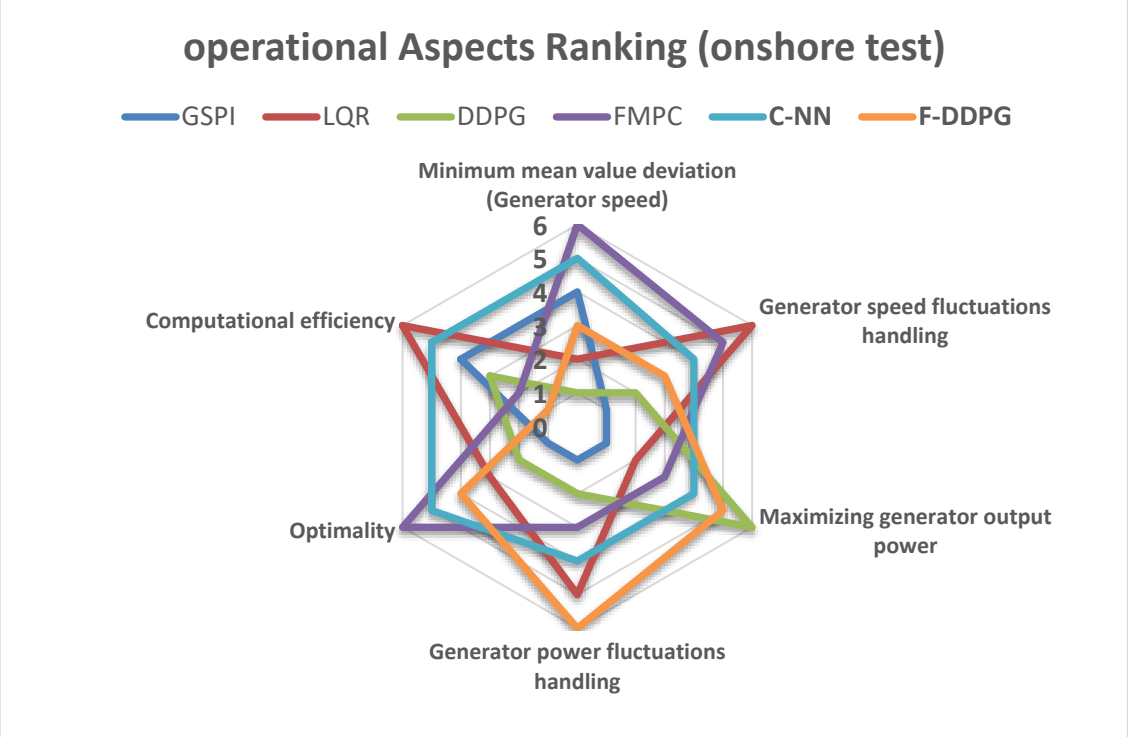
7. Computational Efficiency:

   - This measures the computational resources required by the controller to perform its functions. More computationally efficient controllers can operate on less powerful hardware and may respond faster to changing conditions, which is important for real-time control applications.

The spider radar chart is used to perform such comparison. The values indicated on the charts varies from one to six. One means the lowest rank and six means the highest rank. Figures (4.19-a) and (4.19-b) represent the ranking of different controllers in both offshore and onshore tests, respectively.



(a)



(b)

Figure 4.19 The spider radar scheme representing the ranking of the controllers regarding operational aspects

(a) Offshore Scenario I (b) Onshore Scenario II

Chapter 5 Discussion and Conclusions

5.1. Discussion

The proposed CNN and F-DDPG controllers show promising performance in effective collective pitch control for WT systems. Their efficacy is benchmarked against distinct baseline controllers: GSPI, LQR, DDPG, and FMPC each exhibiting unique strengths and weaknesses.

GSPI excels in keeping the generator speed nearly at its rated value but falls short in reducing fluctuations and enhancing output power. The LQR, an optimal controller designed at a reduced-order WT system, excels in reducing fluctuations and ensuring stability, particularly in onshore conditions. However, its effectiveness decreases in offshore models with added hydrodynamic challenges and external wave perturbations. DDPG, an RL controller, is trained at a median wind speed of 18 m/s, within the wind speed range (12 m/s to 24 m/s). While it maximizes output power in moderately turbulent onshore winds, it struggles to maintain rated generator speed and faces increased fluctuations and diminished power generation in highly turbulent offshore conditions. The FMPC is a model-based controller designed at a reduced-order onshore WT system. Despite showing superior performance in onshore conditions, its performance degrades in offshore conditions. Also, it encounters low computational efficiency, recording longer simulation time than GSPI, LQR, DDPG and CNN-Cs.

The proposed CNN-C is evolved from the FMPC being operating on onshore WT. Thus, its performance is almost identical to the FMPC at the case of the onshore conditions, achieving great score of optimality like FMPC. However, its main strength points are the performance generalization capability and the computational efficiency. As inferred from the offshore test, the proposed CNN-C achieved significantly greater optimality index than the FMPC. In addition, it records the lowest simulation time in both test scenarios. The strengths are highlighted by the enhancement ratios of the proposed C-NN controller compared to others, as detailed in Table 4.3 in Subsection 4.4.2.

The proposed F-DDPG controller marks a significant improvement in various areas, effectively maintaining generator speed near its rated value in both moderate and highly turbulent profiles, thanks to the fuzzy system which smooths the DDPG agents' transitions, thereby enhancing the controller's robustness. It also reduces generator speed and power fluctuations, adeptly manages new and complex hydrodynamics, including external wave and current disturbances, and achieves greater optimality. The main drawback of this method is the computational efficiency as it recorded the longest simulation interval in both test cases. The strengths are highlighted by the enhancement ratios of the proposed F-DDPG controller compared to others, as detailed in Table 4.4 in Subsection 4.4.2.

The proposed CNN-C main strengths in terms of design and operation could be summarized as follows:

1. The CNN-C is evolved from the supervised learning concept and the FMPC on significantly low number of samples.
2. The neural network topology is very simple, direct to understand and has few numbers of neurons. This ensures the simplicity of the design.
3. The training data encompass the primary states that affect the controller's performance and efficient samples that cover all region 3 operation range. Thus, there is no need to include the fuzzy logic system that manipulates the system parameters being fed to the controller, as in the case of the FMPC.
4. The training time required to train the proposed controller is significantly low (just 8 minutes only), as mentioned in the design section of this method.
5. The main contribution of this method is the significant highest computational efficiency, as the proposed controller has recorded the shortest simulation time in comparison to other controllers.
6. The proposed method represents significant enhancement in the performance and performance generalization as compared to the FMPC in the case of offshore test.

The proposed F-DDPG controller's main strengths in terms of design and operation could be summarized as follows:

1. Imitation learning is used to provide the RL agent with highly efficient samples during the initial training episodes, which helps in faster convergence, reaching a consistent optimal level of training.
2. The actor-critic networks topology modification results in a reduction of the training computational cost by a factor of 1.6 from the standard DDPG networks topology.
3. The robustness of the controller is ensured by measuring the effect of wind inflow perturbations and uncertainties on varying the system dynamics using the gap-metric criterion.
4. The fuzzy system aids in providing smooth transitions and blending the control actions of different DDPG agents, achieving optimal, stable, and robust operation.
5. The proposed controller is trained in a medium-fidelity onshore environment. Still, it has the capability to generalize its performance in newly unconsidered complex hydrodynamics and external conditions (waves and currents), as shown in the case of the high-fidelity offshore model test.
6. The F-DDPG controller scores the highest optimality value, indicating its superiority in performance over all other controllers, especially in the offshore conditions.

Although the proposed CNN-C has its strengths, it presents some limitations. These limitations could be summarized as follows:

1- The proposed method is built on function approximation using CNNs. Thus, no stability proof is guaranteed in this method, as this is one of the challenging parts of AI-based controllers.
2- The proposed controller is not an optimal controller, but evolved from an optimal one (FMPC). This is because there is no objective function (reward function or cost function) to address the operational requirements, unlike the proposed F-DDPG controller. Instead, the neural network parameters update

relies on minimizing the squared error between the FMPC control action and the proposed controller control action.

3- The validity of the proposed controller is ensured through severe validation tests at different conditions to compensate the need for stability analysis.
4- The proposed method is just mimic the behavior of the FMPC without any intended approach in the design to enhance the performance over the FMPC, unlike the proposed F-DDPG controller.

While effective, the proposed F-DDPG controller presents certain limitations. These limitations could be summarized as follows:

1- The training relies on imitation learning by collecting samples from a model-based FMPC controller. The design of the FMPC necessitates a reduced-order WT system model, which could pose technical challenges in systems lacking explicit models.
2- Formulating the reward function requires a high understanding of the desirable outcome. In this study, the reward function is formulated to be logic-based, which achieves the requirements for maintaining generator speed and power close to rated values. However, this type of reward function could be challenging in other applications.
3- The DDPG training relies on the bellman equation, and the truncation of the control action during the training could limit the exploration. Due to the insights gained from the optimal control action exerted by the FMPC, we know the optimal region of control action exploration.
4- The reinforcement learning controllers need severe testing and validation on real-world applications to compensate for the stability proof, which is a challenging research topic. To address this issue, simulations on high-fidelity onshore and offshore cases validate the proposed controller performance to ensure its optimality and stability.
5- The computational efficiency in this method is relatively low, unlike the proposed CNN-C.

An enormous number of control processes are subjected to the proposed F-DDPG approach. Examples illustrating the prospective systems in which this approach could be used are shown in Table 5.1. In each of these systems, the key characteristics that align with our approach include the need for real-time adaptive control, dealing with stochastic (random and unpredictable) environmental or system conditions. The combination of fuzzy logic systems and RL agents provides a powerful toolset for such scenarios, offering the flexibility to handle model uncertainty (through fuzzy logic) and the ability to learn optimal control strategies in complex, multidimensional spaces (via DDPG). In addition, any legacy controller applicable to such environments could act as a demonstrator for imitation learning in the initial learning episodes, which could speed up the convergence of the training.

The intended future work will treat all these issues and provide stability analysis proof for the supervised and RL controllers.

Table 5.1 Prospective applications subjected to the proposed F-DDPG approach.

Environment	Prospective observations	Prospective reward function	Prospective target
Autonomous Vehicle Systems	Position, velocity, acceleration, proximity to obstacles	Safe navigation, minimal traffic rule violations, and efficient path planning.	Proposed controller could handle complex driving scenarios, nonlinear vehicle dynamics, and unpredictable road conditions.
Robotic Manipulators in Manufacturing	Joint angles, velocities, torque, end-effector position, and object properties	Precision in object manipulation, energy efficiency, speed of operation, and safety	The proposed controller is useful in controlling robotic arms with complex, nonlinear dynamics and requiring precision in dynamic environments.
Smart Grid Energy Management	Energy demand, supply levels, grid stability metrics, and stochastic weather conditions	Balance between supply and demand, grid stability, cost-effectiveness, and renewable energy utilization.	The proposed controller could manage the variable nature of renewable energy sources and fluctuating demand.

5.2. Conclusions

The thesis has proposed two novel model-free AI-based collective pitch angle controllers. The objective of the CPC is to regulate the generator speed and power at their rated values with minimum possible fluctuations. This is achieved mainly in region 3 (constant power region), where wind speeds are relatively high (rated speed: 12 m/s up to cut-out speed: 25 m/s).

The first proposed method in this thesis has suggested a novel AI-based CPC controller based on supervised learning and neural networks. A FMPC is used to provide high efficiency training samples through operating it in six different operating points. The CNN is selected due to its capability to capture the linear and nonlinear relations between the inputs and outputs. Being of relatively small number of neurons (50 neurons), the training time is noticed to be significantly low as it does not exceed 9 minutes. The training process of the CNN started with forwarding the state inputs of the system to exert the estimated CPC output. Then, loss function is constructed by measuring the sum of square errors between the estimated CPC and actual CPC from the FMPC. Afterwards, the LMB algorithm is used to optimize the loss function and update the network weights. Following that, the tunning of hyperparameters is delivered. This is achieved through different trials of changing the number of layers, neurons and learning rate then retrain the model and observe the value of the loss function. It is noticed that four hidden layers with (20,15,10,5) neurons is the optimal choice as this suggested topology achieved the least value of loss function during training and testing conditions. At the deployment stage, the proposed CNN-C controller's performance is compared to GSPI, and FMPC controllers. The results have

been validated and generalized on two case scenarios: high-fidelity 5-MW onshore and offshore models. The CNN-C controller's performance results have indicated the generalization and robustness nature of performance, particularly in the offshore test where the challenging hydrodynamics and external conditions of waves have been activated. A significant improvement over its predecessor FMPC has been noticed in generator output power, energy extraction, reduction in fluctuations, and optimality especially in the offshore conditions. Also, the most notably improvement is in the computational complexity which has been significantly reduced compared to all other controllers.

The second proposed method in this thesis has suggested a novel collective pitch angle controller design depending on hybrid imitation learning and model-free RL techniques. Six DDPG agents have been trained at different wind speed profiles with averages covering region 3. The agents have been trained in a medium-fidelity environment to enable the onshore WT to have degrees of freedom. In the initial training phase, the FMPC has served as a guide for imitation learning, collecting highly effective samples for storage in the experience replay buffer. These samples have allowed the agents to initially adopt the characteristics of the FMPC. Subsequently, DDPG agents have taken over to learn through direct interaction with the environment to achieve better cumulative rewards than FMPC. This hybrid technique has fastened the training, which has converged after approximately 25 episodes. The actor-critic networks topology has reduced the training computational time by 1.6 while deepening the complex patterns learned. The actor network has been trained to maximize the cumulative estimated return. The critic network has been trained to minimize the difference between the estimated and the target returns to track the actual reward's behavior. The logic-based reward function has achieved the objective of maintaining the operation of WT at rated generator speed and power. During the deployment phase, a fuzzy system has processed the six best-trained agents to form the optimal CPC action and ensure controller robustness under different operating conditions. The proposed F-DDPG controller's performance is compared to that of GSPI, LQR, and single-DDPG-agent controllers. The results have been validated and generalized on two case scenarios: high-fidelity 5-MW onshore and offshore models. The F-DDPG controller's performance results have indicated the generalization and robustness nature of performance, particularly in the offshore test where the challenging hydrodynamics and external conditions of waves have been activated. A significant improvement has been noticed in generator output power, energy extraction, reduction in fluctuations, and optimality. However, the computational complexity observed is relatively high compared to other controllers.

The simulations and results have been conducted on the Open-FAST package integrated with MATLAB/Simulink. A comprehensive numerical analysis, conducted on the high-fidelity OpenFAST 5-MW WT model, has been provided to show the significance in performance of the proposed methods in all operating points of region 3.

5.3. Future Work

The research on high-reliable, robust, high-performance, computational-efficient, AI-based controllers in the field of WTs is still a vast horizon. Various aspects may be incorporated and considered in pitch control design to enhance performance. These aspects are:

1- Using a dedicated deep RL-based algorithms which include constraints handling like Proximal policy optimization (PPO) and trust region policy optimization (TRPO) algorithms.

2- The revolutionary application of the recent reinforcement learning from human feedback (RLHF) in the field.

3- The studying of the wake effect along with an efficient CPC. The wake effect refers to the reduction in wind speed and the increase in turbulence intensity that occurs downstream of a wind turbine. When a turbine extracts kinetic energy from the wind to generate electricity, it creates a region of slower-moving, more turbulent air behind it, known as the wake. This can negatively impact the performance of downstream turbines in a wind farm, leading to reduced power output and increased mechanical stress on those turbines.

4- The challenging stability analysis of such AI-based controllers.

References

[1] I. Renewable Energy Agency, “RENEWABLE ENERGY STATISTICS 2023 STATISTIQUES D’ÉNERGIE RENOUVELABLE 2023 ESTADÍSTICAS DE ENERGÍA RENOVABLE 2023 About IRENA,” 2023. Accessed: Jan. 20, 2024. [Online]. Available: https://www.irena.org/Publications/2023/Jul/Renewable-energy-statistics-2023

[2] Global Wind Energy Council, “GWEC-2023_interactive,” 2023. Accessed: Jan. 20, 2024. [Online]. Available: https://gwec.net/globalwindreport2023/

[3] J. F. Manwell, J. G. McGowan, and A. L. Rogers, *Wind Energy Explained: Theory, Design and Application*, 2. Aufl. New York: Wiley, 2010. doi: 10.1002/9781119994367.

[4] T. Burton, D. Sharpe, N. Jenkins, and E. Bossanyi, “The Wind Energy Handbook,” vol. 1, 2001. doi: 10.1002/0470846062.

[5] J. Xu, J. Wang, J. Rao, Y. Zhong, and H. Wang, “Adaptive dynamic programming for optimal control of discrete-time nonlinear system with state constraints based on control barrier function,” *International Journal of Robust and Nonlinear Control*, vol. 32, no. 6, pp. 3408–3424, 2022, doi: https://doi.org/10.1002/rnc.5955.

[6] J. Jonkman, “OpenFAST Documentation.” Accessed: Aug. 24, 2023. [Online]. Available: https://openfast.readthedocs.io/en/main/

[7] K. Johnson, L. Pao, M. Balas, and L. Fingersh, “Control of variable-speed wind turbines: Standard and adaptive techniques for maximizing energy capture,” *Control Systems, IEEE*, vol. 26, pp. 70–81, Aug. 2006, doi: 10.1109/MCS.2006.1636311.

[8] A. Lasheen and A. L. Elshafei, “Wind-turbine collective-pitch control via a fuzzy predictive algorithm,” *Renew Energy*, vol. 87, pp. 298–306, Mar. 2016, doi: 10.1016/j.renene.2015.10.030.

[9] D. Kumar and K. Chatterjee, “A review of conventional and advanced MPPT algorithms for wind energy systems,” *Renewable and Sustainable Energy Reviews*, vol. 55, pp. 957–970, Mar. 2016, doi: 10.1016/J.RSER.2015.11.013.

[10] Q. Wang and L. Chang, “An intelligent maximum power extraction algorithm for inverter-based variable speed wind turbine systems,” *IEEE Trans Power Electron*, vol. 19, no. 5, pp. 1242–1249, 2004, doi: 10.1109/TPEL.2004.833459.

[11] K. Ramireddy, H. Harsh, Y. V Pavan Kumar, D. Pradeep, P. Reddy Ch, and R. Kannan, “Modelling of Neural Network-based MPPT Controller for Wind Turbine Energy System,” in *Control and Measurement Applications for Smart*

Grid, vol. 822, Singapore: Springer, 2022, pp. 429–439. doi: 10.1007/978-981-16-7664-2_35.

[12] A. Mesemanolis and C. Mademlis, "A Neural Network Based MPPT Controller for Variable Speed Wind Energy Conversion Systems," in *IET Conference Publications*, 8th Mediterranean Conference on Power Generation, Transmission, Distribution and Energy Conversion (MEDPOWER 2012), Mar. 2012, pp. 1–6. doi: 10.1049/cp.2012.2034.

[13] E. Muñoz Palomeque, J. Sierra-García, and M. Santos Peñas, "MPPT Control in an Offshore Wind Turbine Optimized with Genetic Algorithms and Unsupervised Neural Networks," in *Artificial Intelligence Applications and Innovations*, vol. 676, Cham: Springer, 2023, pp. 465–477. doi: 10.1007/978-3-031-34107-6_37.

[14] M.-F. Tsai, C.-S. Tseng, and Y.-H. Hung, "A novel MPPT control design for wind-turbine generation systems using neural network compensator," in *IECON Proceedings (Industrial Electronics Conference)*, Mar. 2012, pp. 3521–3526. doi: 10.1109/IECON.2012.6389333.

[15] M. J. Khan, "An AIAPO MPPT controller based real time adaptive maximum power point tracking technique for wind turbine system," *ISA Trans*, vol. 123, pp. 492–504, 2022, doi: https://doi.org/10.1016/j.isatra.2021.06.008.

[16] C. Wei, Z. Zhang, W. Qiao, and L. Qu, "Reinforcement-Learning-Based Intelligent Maximum Power Point Tracking Control for Wind Energy Conversion Systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 10, pp. 6360–6370, 2015, doi: 10.1109/TIE.2015.2420792.

[17] S. S. Shuvo, M. M. Islam, and Y. Yilmaz, "DROP: Deep Reinforcement Learning Based Optimal Perturbation for MPPT in Wind Energy," in *2022 North American Power Symposium (NAPS)*, 2022, pp. 1–6. doi: 10.1109/NAPS56150.2022.10012250.

[18] D. Belkhiri, M. Ajaamoum, K. Cherifi, A. Elidrissi, and M. R. El Moutawakil Alaoui, "Artificial Intelligence-based MPPT Techniques in Wind Energy Systems: A Literature Review," in *2023 3rd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, 2023, pp. 1–6. doi: 10.1109/IRASET57153.2023.10153057.

[19] P. Bagheri, L. Behjat, and Q. Sun, "Nonlinear control of a class of non-affine variable-speed variable-pitch wind turbines with radial-basis function neural networks," *ISA Trans*, vol. 131, pp. 197–209, 2022, doi: https://doi.org/10.1016/j.isatra.2022.05.004.

[20] A. Parvaresh, S. Abrazeh, S.-R. Mohseni, M. J. Zeitouni, M. Gheisarnejad, and M.-H. Khooban, "A Novel Deep Learning Backstepping Controller-Based

Digital Twins Technology for Pitch Angle Control of Variable Speed Wind Turbine," *Designs (Basel)*, vol. 4, no. 2, 2020, doi: 10.3390/designs4020015.

[21] P. Chen, D. Han, F. Tan, and J. Wang, "Reinforcement-based robust variable pitch control of wind turbines," *IEEE Access*, vol. 8, pp. 20493–20502, 2020, doi: 10.1109/ACCESS.2020.2968853.

[22] J. E. Sierra-Garcia and M. Santos, "Combination of Neural Networks and Reinforcement Learning for Wind Turbine Pitch Control," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, 2022, pp. 385–392. doi: 10.1007/978-3-031-15471-3_33.

[23] S. Qin, Y. Liu, Z. Liu, and M. Sun, "Data-based Reinforcement Learning with Application to Wind Turbine Pitch Control," in *2021 6th International Conference on Power and Renewable Energy (ICPRE)*, 2021, pp. 538–542. doi: 10.1109/ICPRE52634.2021.9635193.

[24] J. Xie, H. Dong, and X. Zhao, "Data-driven torque and pitch control of wind turbines via reinforcement learning," *Renew Energy*, vol. 215, Oct. 2023, doi: 10.1016/j.renene.2023.06.014.

[25] J. E. Sierra-Garcia, M. Santos, and R. Pandit, "Wind turbine pitch reinforcement learning control improved by PID regulator and learning observer," *Eng Appl Artif Intell*, vol. 111, May 2022, doi: 10.1016/j.engappai.2022.104769.

[26] N. Tomin, "Robust Reinforcement Learning-Based Multiple Inputs and Multiple Outputs Controller for Wind Turbines," *Mathematics*, vol. 11, no. 14, Jul. 2023, doi: 10.3390/math11143242.

[27] J. E. Sierra-Garcia and M. Santos, "Deep learning and fuzzy logic to implement a hybrid wind turbine pitch control," *Neural Comput Appl*, vol. 34, no. 13, pp. 10503–10517, Jul. 2022, doi: 10.1007/s00521-021-06323-w.

[28] J. E. Sierra-García and M. Santos, "Improving Wind Turbine Pitch Control by Effective Wind Neuro-Estimators," *IEEE Access*, vol. 9, pp. 10413–10425, 2021, doi: 10.1109/ACCESS.2021.3051063.

[29] S. Wu, Y. Wang, and S. Cheng, "Extreme learning machine based wind speed estimation and sensorless control for wind turbine power generation system," *Neurocomputing*, vol. 102, pp. 163–175, 2013, doi: https://doi.org/10.1016/j.neucom.2011.12.051.

[30] J. Xie, H. Dong, and X. Zhao, "Power Regulation and Load Mitigation of Floating Wind Turbines via Reinforcement Learning," *IEEE Transactions on Automation Science and Engineering*, 2023, doi: 10.1109/TASE.2023.3295576.

[31] B. Fernandez-Gauna, M. Graña, J. L. Osa-Amilibia, and X. Larrucea, "Actor-critic continuous state reinforcement learning for wind-turbine control robust

optimization," *Inf Sci (N Y)*, vol. 591, pp. 365–380, Apr. 2022, doi: 10.1016/j.ins.2022.01.047.

[32] Q. Bin, L. Pengcheng, W. Xin, and Z. Wanli, "Pitch angle control based on renforcement learning," in *The 26th Chinese Control and Decision Conference (2014 CCDC)*, 2014, pp. 18–21. doi: 10.1109/CCDC.2014.6852110.

[33] J. E. Sierra-García and M. Santos, "Exploring reward strategies for wind turbine pitch control by reinforcement learning," *Applied Sciences (Switzerland)*, vol. 10, no. 21, pp. 1–23, Nov. 2020, doi: 10.3390/app10217462.

[34] E. Hosseini, E. Aghadavoodi, and L. M. Fernández Ramírez, "Improving response of wind turbines by pitch angle controller based on gain-scheduled recurrent ANFIS type 2 with passive reinforcement learning," *Renew Energy*, vol. 157, pp. 897–910, 2020, doi: https://doi.org/10.1016/j.renene.2020.05.060.

[35] Z. Cao, A. Yazdani, and A. Mahmoudi, "Intelligent robust pitch control of wind turbine using brain emotional learning," *International Transactions on Electrical Energy Systems*, vol. 31, Mar. 2021, doi: 10.1002/2050-7038.12785.

[36] A. S. Yilmaz and Z. Özer, "Pitch angle control in wind turbines above the rated wind speed by multi-layer perceptron and radial basis function neural networks," *Expert Syst Appl*, vol. 36, no. 6, pp. 9767–9775, 2009, doi: https://doi.org/10.1016/j.eswa.2009.02.014.

[37] I. Poultangari, R. Shahnazi, and M. Sheikhan, "RBF neural network based PI pitch controller for a class of 5-MW wind turbines using particle swarm optimization algorithm," *ISA Trans*, vol. 51, pp. 641–648, Mar. 2012, doi: 10.1016/j.isatra.2012.06.001.

[38] J. Du and B. Wang, "Pitch Control of Wind Turbines Based on BP Neural Network PI," *J Phys Conf Ser*, vol. 1678, p. 12060, Mar. 2020, doi: 10.1088/1742-6596/1678/1/012060.

[39] A. Najd, G. Gorel, and H. Hammood, "Pitch angle control using neural network in wind turbines," *IOP Conf Ser Mater Sci Eng*, vol. 928, p. 22118, Mar. 2020, doi: 10.1088/1757-899X/928/2/022118.

[40] W. Jie, C. Jingchun, Y. Lin, W. Wenliang, and D. Jian, "Pitch control of wind turbine based on deep neural network," *IOP Conf Ser Earth Environ Sci*, vol. 619, p. 12034, Mar. 2020, doi: 10.1088/1755-1315/619/1/012034.

[41] X. Jiao, W. Meng, Q. Yang, L. Fu, and Q. Chen, "Adaptive Continuous Neural Pitch Angle Control for Variable-Speed Wind Turbines," *Asian J Control*, vol. 21, Mar. 2018, doi: 10.1002/asjc.1963.

[42] J. E. Sierra-García and M. Santos, "Lookup Table and Neural Network Hybrid Strategy for Wind Turbine Pitch Control," *Sustainability*, vol. 13, no. 6, 2021, doi: 10.3390/su13063235.

[43] R. Sitharthan, K. R. Devabalaji, and A. Jees, “An Levenberg–Marquardt trained feed-forward back-propagation based intelligent pitch angle controller for wind generation system,” *Renewable Energy Focus*, vol. 22–23, pp. 24–32, 2017, doi: https://doi.org/10.1016/j.ref.2017.10.003.

[44] A. Dahbi, N. Nait-Said, and M.-S. Nait-Said, “A novel combined MPPT-pitch angle control for wide range variable speed wind turbine based on neural network,” *Int J Hydrogen Energy*, vol. 41, no. 22, pp. 9427–9442, 2016, doi: https://doi.org/10.1016/j.ijhydene.2016.03.105.

[45] H. Dastres, A. Mohammadi, and M. Shamekhi, “A Neural Network Based Adaptive Sliding Mode Controller for Pitch Angle Control of a Wind Turbine,” in *2020 11th Power Electronics, Drive Systems, and Technologies Conference (PEDSTC)*, 2020, pp. 1–6. doi: 10.1109/PEDSTC49159.2020.9088373.

[46] Z. Wang, C. Cai, and K. Jia, “Neural network adaptive control for constant output power of variable pitch wind turbine,” in *Proceedings of 2013 IEEE International Conference on Vehicular Electronics and Safety*, 2013, pp. 165–170. doi: 10.1109/ICVES.2013.6619623.

[47] Z. Wang, Z. Shen, C. Cai, and K. Jia, “Adaptive control of wind turbine generator system based on RBF-PID neural network,” in *2014 International Joint Conference on Neural Networks (IJCNN)*, 2014, pp. 538–543. doi: 10.1109/IJCNN.2014.6889538.

[48] C.-H. Chen, C.-M. Hong, and T.-C. Ou, “Hybrid fuzzy control of wind turbine generator by pitch control using RNN,” *International Journal of Ambient Energy*, vol. 33, pp. 56–64, 2012, [Online]. Available: https://api.semanticscholar.org/CorpusID:108683491

[49] S. A. Raza and A. H. M. Abdur Rahim, “A Differential Evolution Based Adaptive Neural Network Pitch Controller for a Doubly Fed Wind Turbine Generator System,” *Research Journal of Applied Sciences, Engineering and Technology*, vol. 6, pp. 4271–4280, Mar. 2013, doi: 10.19026/rjaset.6.3544.

[50] J. E. Sierra-García and M. Santos, “Performance Analysis of a Wind Turbine Pitch Neurocontroller with Unsupervised Learning,” *Complexity*, vol. 2020, p. 4681767, 2020, doi: 10.1155/2020/4681767.

[51] B. Warsito, R. Santoso, Suparti, and H. Yasin, “Cascade Forward Neural Network for Time Series Prediction,” *J Phys Conf Ser*, vol. 1025, no. 1, p. 12097, May 2018, doi: 10.1088/1742-6596/1025/1/012097.

[52] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” Sep. 2015, [Online]. Available: http://arxiv.org/abs/1509.02971

[53] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction, 2nd ed.* in Adaptive computation and machine learning. Cambridge, MA, US: The MIT Press, 2018.

[54] H.-S. Yan and G.-B. Wang, "Adaptive tracking control for stochastic nonlinear systems with time-varying delays using multi-dimensional Taylor network," *ISA Trans*, vol. 132, pp. 246–257, 2023, doi: https://doi.org/10.1016/j.isatra.2022.06.004.

[55] X. Jin, H. Ma, J. Tang, and Y. Kang, "A Self-Adaptive Vibration Reduction Method Based on Deep Deterministic Policy Gradient (DDPG) Reinforcement Learning Algorithm," *Applied Sciences (Switzerland)*, vol. 12, no. 19, Oct. 2022, doi: 10.3390/app12199703.

[56] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[57] L. Buşoniu, T. De Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annu Rev Control*, vol. 46, pp. 8–28, 2018, doi: https://doi.org/10.1016/j.arcontrol.2018.09.005.

[58] Q. Zou, K. Xiong, and Y. Hou, "An end-to-end learning of driving strategies based on DDPG and imitation learning," in *2020 Chinese Control And Decision Conference (CCDC)*, 2020, pp. 3190–3195. doi: 10.1109/CCDC49329.2020.9164410.

[59] H. Xie, X. Xu, Y. Li, W. Hong, and J. Shi, "Model Predictive Control Guided Reinforcement Learning Control Scheme," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8. doi: 10.1109/IJCNN48605.2020.9207398.

[60] J. Fang *et al.*, "Wind turbine rotor speed design optimization considering rain erosion based on deep reinforcement learning," *Renewable and Sustainable Energy Reviews*, vol. 168, p. 112788, 2022, doi: https://doi.org/10.1016/j.rser.2022.112788.

[61] D. S. N. J. E. B. Tony Burton, *WIND ENERGY HANDBOOK*. John Wiley & Sons, 2001. doi: 10.1002/9781119992714.

[62] H. Bailey, K. Brookes, and P. Thompson, "Assessing Environmental Impacts of Offshore Wind Farms: Lessons Learned and Recommendations for the Future," *Aquat Biosyst*, vol. 10, p. 8, Mar. 2014, doi: 10.1186/2046-9063-10-8.

[63] S. Eriksson, H. Bernhoff, and M. Leijon, "Evaluation of different turbine conceps for wind power," *Renewable and Sustainable Energy Reviews*, vol. 12, pp. 1419–1434, Mar. 2008, doi: 10.1016/j.rser.2006.05.017.

[64] X. and J. H. and D. D. Solomin E. and Lingjie, “Comprehensive Comparison of the Most Effective Wind Turbines,” in *Advances in Automation*, A. S. Radionov Andrey A. and Karandaev, Ed., Cham: Springer International Publishing, 2020, pp. 588–595.

[65] M. Ba Alawi, “The integration of wind turbines for generating sustainable energy in skyscrapers,” Mar. 2018.

[66] Courtney Powell, “Electricity from the Wind – Part 1,” xenogyre. Accessed: Mar. 23, 2024. [Online]. Available: https://xenogyre.com/2020/12/16/electricity-from-wind-turbines-part-1/

[67] V. s. K. V. Harish and A. Sant, “Grid Integration of Wind Energy Conversion Systems,” 2020. doi: 10.1007/698_2020_610.

[68] J. Baran and A. Jaderko, “An MPPT Control of a PMSG-Based WECS with Disturbance Compensation and Wind Speed Estimation,” *Energies (Basel)*, vol. 13, p. 6344, Mar. 2020, doi: 10.3390/en13236344.

[69] M. Abdelateef, E. Abd El-Hay, and M. M. Elkholy, “Recent Trends in Wind Energy Conversion System with Grid Integration Based on Soft Computing Methods: Comprehensive Review, Comparisons and Insights,” *Archives of Computational Methods in Engineering*, vol. 30, Mar. 2022, doi: 10.1007/s11831-022-09842-4.

[70] J. Jonkman, “Definition of the Floating System for Phase IV of OC3,” 2010. [Online]. Available: http://www.osti.gov/bridge

[71] J. Jonkman, “The New Modularization Framework for the FAST Wind Turbine CAE Tool Preprint,” 2013. [Online]. Available: http://www.osti.gov/bridge

[72] S. Dhar, “Development and Validation of Small-scale Model to Predict Large Wind Turbine Behavior,” 2006.

[73] F. Tamarit, E. García, E. Quiles, and A. Correcher, “Model and Simulation of a Floating Hybrid Wind and Current Turbines Integrated Generator System, Part I: Kinematics and Dynamics,” *J Mar Sci Eng*, vol. 11, no. 1, 2023, doi: 10.3390/jmse11010126.

[74] G. S. Bir, “User’s Guide to MBC3 (Multi-blade Coordinate Transformation Utility for 3-Bladed Wind Turbines),” 2008.

[75] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., in Proceedings of Machine Learning Research, vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, Mar. 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[76] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” *CoRR*, vol. abs/1502.01852, 2015, [Online]. Available: http://arxiv.org/abs/1502.01852

[77] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[78] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley & Sons, Inc., 1994.

[79] ChessJournal, “How Many Possible Moves Are There In Chess?” Accessed: Mar. 24, 2024. [Online]. Available: https://www.chessjournal.com/how-many-possible-moves-are-there-in-chess/

[80] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 1st ed. Athena Scientific, 1995.

[81] Russ Tedrake, “Underactuated Robotics : Linear Quadratic Regulators,” MIT. Accessed: Mar. 24, 2024. [Online]. Available: https://underactuated.mit.edu/lqr.html

[82] Pe Supavish, “Policy Iteration : Easy Example,” Medium. Accessed: Mar. 24, 2024. [Online]. Available: https://medium.com/@pesupavish/policy-iteration-easy-example-d3fd1eb98c6c

[83] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Mach Learn*, vol. 8, no. 3, pp. 279–292, 1992, doi: 10.1007/BF00992698.

[84] J. Peters and J. A. Bagnell, “Policy Gradient Methods,” Mar. 2018, doi: 10.1184/R1/6558227.v1.

[85] Jihao Long and Jiequn Han, “Reinforcement Learning with Function Approximation: From Linear to Nonlinear,” *Journal of Machine Learning*, vol. 2, no. 3, pp. 161–193, Jun. 2023, doi: 10.4208/jml.230105.

[86] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” 2013.

[87] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic Policy Gradient Algorithms,” in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., in Proceedings of Machine Learning Research, vol. 32. Bejing, China: PMLR, Mar. 2014, pp. 387–395. [Online]. Available: https://proceedings.mlr.press/v32/silver14.html

[88] Ashwin Rao, “Policy Gradient Algorithms,” 2020. Accessed: Mar. 24, 2024. [Online]. Available: https://web.stanford.edu/class/cme241/lecture_slides/PolicyGradient.pdf

[89] A. Lasheen, M. S. Saad, H. M. Emara, and A. L. Elshafei, “Tube-based explicit model predictive output-feedback controller for collective pitching of wind

turbines," *Renew Energy*, vol. 131, pp. 549–562, Feb. 2019, doi: 10.1016/j.renene.2018.07.033.

[90] S.-H. Kim, "Control of direct current motors," in *Electric Motor Control*, Elsevier, 2017, pp. 39–93. doi: 10.1016/b978-0-12-812138-2.00002-7.

[91] Y. Nam, "Control System Design," in *Wind Turbines*, I. Al-Bahadly, Ed., Rijeka: IntechOpen, 2011, ch. 11. doi: 10.5772/15841.

[92] C. Hildreth, "A quadratic programming procedure," *Naval Research Logistics Quarterly*, vol. 4, no. 1, pp. 79–85, 1957, doi: https://doi.org/10.1002/nav.3800040113.

[93] L. and L. H. Refaeilzadeh Payam and Tang, "Cross-Validation," in *Encyclopedia of Database Systems*, M. T. LIU LING and ÖZSU, Ed., Boston, MA: Springer US, 2009, pp. 532–538. doi: 10.1007/978-0-387-39940-9_565.

[94] Andrew Ng et al., "Deep Learning Specialization," Coursera. Accessed: Jun. 02, 2024. [Online]. Available: https://www.coursera.org/specializations/deep-learning

[95] Łukasz Gebel, "Why we need bias in neural networks," towardsdatascience. Accessed: Mar. 28, 2024. [Online]. Available: https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98

[96] S. Sharma, S. Sharma, and A. Athaiya, "ACTIVATION FUNCTIONS IN NEURAL NETWORKS," *International Journal of Engineering Applied Sciences and Technology*, 2020, [Online]. Available: https://api.semanticscholar.org/CorpusID:225922639

[97] J. J. Moré, "The Levenberg-Marquardt algorithm: Implementation and theory," in *Numerical Analysis*, G. A. Watson, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 105–116.

[98] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, Mar. 2011.

[99] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. doi: https://doi.org/10.48550/arXiv.1412.6980.

[100] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT'2010*, G. Lechevallier Yves and Saporta, Ed., Heidelberg: Physica-Verlag HD, 2010, pp. 177–186.

[101] M. T. Hagan and M. B. Menhaj, “Training feedforward networks with the Marquardt algorithm,” *IEEE Trans Neural Netw*, vol. 5, no. 6, pp. 989–993, 1994, doi: 10.1109/72.329697.

[102] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.

[103] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, Aug. 2010.

[104] C. C. Aggarwal, “Neural Networks and Deep Learning,” *Neural Networks and Deep Learning*, 2018, doi: 10.1007/978-3-319-94463-0.

[105] L. Lu, “Dying ReLU and Initialization: Theory and Numerical Examples,” *Commun Comput Phys*, vol. 28, no. 5, pp. 1671–1706, Jun. 2020, doi: 10.4208/cicp.oa-2020-0165.

[106] N. Ashraf, R. Mostafa, R. Sakr, and M. Rashad, “Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm,” *PLoS One*, vol. 16, p. e0252754, Apr. 2021, doi: 10.1371/journal.pone.0252754.

[107] B. He, H. Zhao, G. Liang, J. Zhao, J. Qiu, and Z. Y. Dong, “Ensemble-based Deep Reinforcement Learning for robust cooperative wind farm control,” *International Journal of Electrical Power & Energy Systems*, vol. 143, p. 108406, 2022, doi: https://doi.org/10.1016/j.ijepes.2022.108406.

[108] R. E. Precup, R. C. Roman, and A. Safaei, *Data-driven Model-free Controllers*. CRC Press, 2021. [Online]. Available: https://books.google.com.eg/books?id=yFi2zgEACAAJ

[109] R.-E. Precup *et al.*, “Fuzzy Control System Performance Enhancement by Iterative Learning Control,” *IEEE Transactions on Industrial Electronics*, vol. 55, no. 9, pp. 3461–3475, 2008, doi: 10.1109/TIE.2008.925322.

[110] A. Sadollah, *Fuzzy Logic Based in Optimization Methods and Control Systems and Its Applications*. Rijeka: IntechOpen, 2018. doi: 10.5772/intechopen.73112.

[111] B. J. Jonkman, “Turbsim User’s Guide V2.0,” 2014. [Online]. Available: www.nrel.gov/publications.

[112] European Wind Energy Association, “Wind Energy Fact Sheet.” Accessed: Aug. 24, 2023. [Online]. Available: https://www.ewea.org/wind-energy-basics/facts/

[113] P. E. Morthorst and L. Kitzing, “Economics of building and operating offshore wind farms,” in *Offshore Wind Farms: Technologies, Design and Operation*, Elsevier Inc., 2016, pp. 9–27. doi: 10.1016/B978-0-08-100779-2.00002-7.

[114] Q. Hawari, T. Kim, C. Ward, and J. Fleming, “A robust gain scheduling method for a PI collective pitch controller of multi-MW onshore wind turbines,” *Renew Energy*, vol. 192, pp. 443–455, 2022, doi: https://doi.org/10.1016/j.renene.2022.04.117.

[115] R. M. Imran, D. M. A. Hussain, and M. Soltani, “DAC with LQR control design for pitch regulated variable speed wind turbine,” in *2014 IEEE 36th International Telecommunications Energy Conference (INTELEC)*, 2014, pp. 1–6. doi: 10.1109/INTLEC.2014.6972153.

List of Publications

1- A. Nabeel, A. Lasheen, A. L. Elshafei, and E. Aboul Zahab, “Fuzzy-based collective pitch control for wind turbine via deep reinforcement learning,” *ISA Trans*, vol. 148, pp. 307–325, 2024, doi: https://doi.org/10.1016/j.isatra.2024.03.023.

الملخص

تتقدم هذه الرسالة بمناقشة تصميم طرق مختلفة معتمدة على الذكاء الاصطناعي للتحكم في زوايا شفرات توربينات الرياح العملاقة بشكل جماعي. هذه المتحكمات المقترحة تعمل في نطاق سرعات رياح ما بين سرعة الرياح المقننة التي يتم توليد اقصى قدرة عندها إلى سرعة الرياح التي يتم عندها توقف عمل التربينة (12 م/ث إلى 25 م/ث). الهدف الرئيسي من تصميم المتحكم خلال هذا النطاق هو ضبط سرعة مولدات توربينات الرياح عند السرعة المقننة ليتم توليد القدرة المقننة (5 ميجاوات) وأيضاً تقليل التقلبات الناشئة بسبب عشوائية سرعات الرياح. يتم ذلك عن طريق التحكم اللحظي في زوايا شفرات التربينة بشكل جماعي.

التحكم الأمثل في زوايا شفرات توربينات الرياح يواجه بعض الصعوبات. أهمها هو البحث لإيجاد نموذج رياضي عالي الدقة يأخذ في الاعتبار جميع التغيرات العشوائية اللحظية في سرعات الرياح وتأثيرها على ديناميكيات التوربينات. لذلك تقدم هذه الرسالة تصميم متحكمات آلية باستخدام أفرع مختلفة من أساليب الذكاء الاصطناعي للتغلب على هذه المشكلة. يتم ذلك عن طريق انشاء متحكمات تعتمد على نمذجة رياضية بسيطة وخالية من النمذجة الرياضية المعقدة لهذه التوربينات. يقدم النهج الأولي وحدة تحكم قائمة على الشبكة العصبية المتتالية (CNN-C)، باستخدام نموذج التعلم الخاضع للإشراف لتكوينه. حيث يتم تدريبه عن طريق التقليد باستخدام بيانات يتم الحصول عليها من عمل متحكم تنبؤي مبني على الضبابية (FMPC) تم تصميمه باستخدام نموذج بسيط للتوربينة. تقترح الإستراتيجية اللاحقة وحدة تحكم متدرجة للسياسة الحتمية العميقة (F-DDPG) قائمة على الضبابية، تتضمن عناصر التعلم بالتقليد من (FMPC) والتعلم المعزز العميق في تصميمها. يقوم التعليم المعزز العميق بتحسين الأداء بصورة أكبر من نظيره والتعويض عن الديناميكيات والعشوائيات للتربينة التي لم يتم نمذجتها.

تم اختبار أداء المتحكمات على نموذج لتوربينات عالية الدقة وتحاكي الواقع باستخدام برنامج (فاست و ماتلاب). فاست هو برنامج تم تصميمه بواسطة معمل الطاقة المتجددة في الولايات المتحدة الأمريكية ويتميز بأنه يحاكي توربينات الرياح بشكل واقعي عن طريق أخذ معظم درجات الحرية لأجزاء التوربينات وتأثير الرياح والأمواج على هيكل التوربينة. أثبتت النتائج التفوق في الأداء للمتحكمات المقترحة على نظريتها التقليدية في جميع الظروف. وأثبتت أيضاً تعميم الأداء في الظروف المختلفة، تحسين ثبات المولد عند السرعة المقننة، تحسين انتاج الطاقة، سرعة الاستجابة وتقليل الوقت الحسابي.

يقوم الفصل الأول من هذه الرسالة بتقديم تعريف المشكلة التي تناقشها الرسالة والحافز من حلها. كذلك يقوم هذا الفصل بتقديم مراجعات للأعمال السابقة للباحثين، المساهمة المقدمة في هذه الرسالة، واخيراً نبذة عن الفصول القادمة وتقسيمتها المختلفة.

الفصل الثاني يقوم بشرح عام لأنظمة توربينات الرياح، أنواعها المختلفة، نمذجتها، تحليلها، وبرامج المحاكاة المستخدمة. كذلك يقوم هذا الفصل بشرح الأسس لأساليب الذكاء الاصطناعي المستخدمة في هذه الرسالة كالتعلم تحت الإشراف والتعلم المعزز العميق.

الفصل الثالث يتم فيه شرح وتفصيل أساليب تصميم المتحكمات المستخدمة في التحكم في شفرات الرياح جماعياً. كذلك يقوم بعرض نتائج تدريب الشبكات العصبية للمتحكمات القائمة على الذكاء الاصطناعي والوصول إلى الغرض المنشود من هذا التدريب.

الفصل الرابع يقوم بعرض نتائج المحاكاة التي تمت على المقارنة بين المتحكمات المقترحة والمتحكمات التقليدية السابقة. أيضاً يتم عرض نتائج تحليل رقمية والتأكد من الأداء الأمثل للمتحكمات المقترحة من مختلف الجوانب.

الفصل الخامس يناقش نتائج المحاكاة التي حصلنا عليها، كذلك يقوم بعرض نقاط القوة والضعف لكل متحكم من المتحكمات. يقوم أيضاً هذا الفصل بالتأكيد على معالجة المتحكمات المقترحة لمشكلات المتحكمات التقليدية وذلك عن طريق التأكيد على مطابقة النتائج بالمساهمات في الرسالة. أخيراً، يختتم هذا الفصل بتلخيص أهم النقاط من هذه الدراسة وعرض الأعمال المستقبلية المنشودة في تطوير المتحكمات المقترحة.

مهندس: عبد الحميد نبيل صادق رزق يونس

تاريخ الميلاد: 16 \ 12 \ 1994

الجنسية: مصري

تاريخ التسجيل: 1 \ 3 \ 2020

تاريخ المنح: \ \2024

القسم: هندسة القوى الكهربية

الدرجة: ماجستير العلوم

المشرفون:

ا.د. عصام الدين محمد أبو الذهب

ا.د. عبد اللطيف محمد رجائي الشافعي

أ.م.د. أحمد عبد الناصر احمد لاشين

الممتحنون:

أ.د/ عصام الدين محمد أبو الذهب (المشرف الرئيسي)

أ.د/ عبد اللطيف محمد رجائي الشافعي (المشرف)

أ.م.د/ أحمد عبد الناصر احمد لاشين (المشرف)

أ.د/ خالد علي محمد المتولي (الممتحن الداخلي)

أ.د/ هاني محمد حسنين محمد (الممتحن الخارجي)

(أستاذ بقسم هندسة القوى الكهربية بجامعة عين شمس)

عنوان الرسالة:

أداء أمثل لتوربينات الرياح من خلال التحكم جماعياً في زوايا الشفرات بإستخدام الذكاء الاصطناعي

الكلمات الدالة:

التحكم في زاوية الشفرة، توربينات الرياح، التعلم المعزز، المنطق الضبابي، الشبكات العصبية

ملخص الرسالة:

تقدم هذه الأطروحة تطويرًا ودراسة تقييمية لاستراتيجيتين جديدتين للتحكم الجماعي في زوايا شفرات تربينات الرياح. تعتمد كل منهما على الذكاء الاصطناعي وخالية من النمذجة الرياضية المعقدة لهذه التوربينات. تهدف هذه المتحكمات إلى تحقيق استقرار سرعة المولد وإخراج الطاقة عند القيم المقننة، وتقليل التقلبات، وخفض النفقات الحسابية، وتحسين الأداء الأمثل للنظام بشكل عام عند سرعات الرياح العالية فوق المقننة. يقدم النهج الأولي وحدة تحكم قائمة على الشبكة العصبية المتتالية (CNN-C)، باستخدام نموذج التعلم الخاضع للإشراف لتكوينه. تقترح الإستراتيجية اللاحقة وحدة تحكم متدرجة للسياسة الحتمية العميقة قائمة على الضبابية (F-DDPG)، تتضمن عناصر التعلم بالتقليد والتعلم المعزز العميق في تصميمها. يتم إجراء اختبارات المحاكاة باستخدام أدوات المحاكاة (فاست و ماتلاب) عالية الدقة وذلك للتأكيد على التفوق في الأداء مقارنة بالمتحكمات التقليدية.

أداء أمثل لتوربينات الرياح من خلال التحكم جماعياً في زوايا الشفرات بإستخدام الذكاء الاصطناعي

اعداد

عبد الحميد نبيل صادق رزق يونس

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة القوى الكهربية

يعتمد من لجنة الممتحنين:

الاستاذ الدكتور: عصام الدين محمد أبو الذهب **المشرف الرئيسي**

الاستاذ الدكتور: عبد اللطيف محمد رجائي الشافعي **مشرف**

الاستاذ مساعد دكتور: أحمد عبد الناصر أحمد لاشين **مشرف**

الاستاذ الدكتور: خالد علي محمد المتولي **الممتحن الداخلي**

الاستاذ الدكتور: هاني محمد حسنين محمد **الممتحن الخارجي**

(أستاذ بقسم هندسة القوى الكهربية بجامعة عين شمس)

كليــة الهندســة - جامعــة القاهــرة

الجيـزة - جمهوريـة مصـرالعربيــة

2024

أداء أمثل لتوربينات الرياح من خلال التحكم جماعياً في زوايا الشفرات بإستخدام الذكاء الاصطناعي

اعداد

عبد الحميد نبيل صادق رزق يونس

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة القوى الكهربية

تحت اشراف

أ.د/ عصام الدين محمد أبو الذهب
.....
قسم القوى الكهربية
كلية الهندسة – جامعة القاهرة

أ.د / عبد اللطيف محمد رجائي الشافعي
.....
قسم القوى الكهربية
كلية الهندسة – جامعة القاهرة

أ.م. د/ أحمد عبد الناصر أحمد لاشين
.....
قسم القوى الكهربية
كلية الهندسة – جامعة القاهرة

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـرالعربيــة
2024

أداء أمثل لتوربينات الرياح من خلال التحكم جماعياً في زوايا الشفرات بإستخدام الذكاء الاصطناعي

اعداد

عبد الحميد نبيل صادق رزق يونس

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة القوى الكهربية

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة
2024