Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

Ella Gardner Øxnevad

Artificial Neural Network-Based Control for Grid-Connected Converter

Master's thesis in Energy and Environmental Engineering
Supervisor: Mohammad Amin
Co-supervisor: Prabhat Ranjan Bana
June 2022

NTNU
Norwegian University of Science and Technology

Ella Gardner Øxnevad

Artificial Neural Network-Based Control for Grid-Connected Converter

Master's thesis in Energy and Environmental Engineering
Supervisor: Mohammad Amin
Co-supervisor: Prabhat Ranjan Bana
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This master's thesis was completed the spring of 2022 as part of the Master of Science program Energy and Environmental Engineering at the Norwegian University of Science and Technology.

I want to thank my supervisor, professor Mohammad Amin at the Department of Electric Power Engineering, for proposing this master's thesis, our weekly meetings and most appreciated feedback. It has been truly inspiring to work with a topic which is at the frontier of urgent challenges.

A huge thanks to my co-supervisor, PhD Candidate Prabhat Ranjan Bana, for your availability and great support. Your help has been invaluable.

Lastly, I would like to thank the two men in my life: Magnus, my soon-to-be husband, and Dave, my father. For keeping my spirits high, for almost always answering the phone and for your endless and indispensable support.

> *The great thing is to last and get your work done and see and hear and learn and understand; and write when there is something that you know; and not before;*
>
> – Ernest Hemingway, *Death in the Afternoon*

Abstract

As the share of renewable energy sources increases, the control methods for grid-connected converters need to be improved to ensure stable and reliable operation. New control methods are replacing conventional proportional plus integral control as they have been shown to be faster, more robust, and more accurate. Among numerous control methods, the use of artificial intelligence, and more specifically the use of artificial neural networks is increasingly drawing attention in research for improved converter control.

This thesis investigates various artificial intelligence technologies for the inner current control for the grid-connected voltage source converter. Amongst these, four different artificial neural network (ANN)-based control methods are designed and investigated. The first controller consists of a feedforward neural network which is trained using simulation data from a conventional PI controller, which is referred to as PI-ANN control. Similarly, the second controller is also based on the feedforward neural network, but in this case it is trained using data from a model predictive controller (MPC-ANN controller). The third controller consists of a recurrent neural network (RNN) which is trained using dynamic programming approximating optimal control, this is referred to as RNN-based control. The fourth and last controller consists of two feedforward neural networks which are trained using direct heuristic dynamic programming (dHDP).

While the RNN-controller needs further improvements to complete the implementation, the other three controllers are fully developed and implemented in the Matlab/Simulink simulation environment. The performance of the PI-ANN, MPC-ANN and dHDP controllers is evaluated and compared, using simulation results obtained from Matlab/Simulink. Whilst the PI-ANN controller shows no improvement compared to the conventional PI controller, the MPC-ANN and the dHDP perform significantly better with less oscillations, improved dynamic response, and robustness against parameter changes. In addition, the performance of the PI, PI-ANN, MPC and the MPC-ANN controllers has been validated through comparison with OPAL RT hardware-in-the-loop simulations.

Sammendrag

Ettersom andelen fornybare energikilder øker, må kontrollmetodene for nett-tilknyttede omformere forbedres for å sikre stabil og pålitelig drift. I forskning for forbedret omformerkontroll rettes det økende oppmerksomhet mot kunstig intelligens-baserte metoder, og særlig bruken av kunstige nevrale nettverk.

Denne oppgaven undersøker ulike teknologier innen kunstig intelligens for strømregulering av en nett-tilkoblet spenningskildeomformer. Blant disse har fire kunstig nevrale nettverk (ANN)-baserte kontrollmetoder blitt designet og undersøkt. Den første regulatoren består av et feedforward nevralt nettverk som trenes ved bruk av simuleringsdata fra en konvensjonell proporsjonal integral (PI)-regulator, kontrollmetoden omtales PI-ANN-regulator. Tilsvarende består også den andre regulatoren av et FFNN, men i dette tilfellet trenes nettverket ved bruk av data fra en modellprediktiv kontroller (MPC), omtalt som MPC-ANN regulator. Den tredje regulatoren består av et tilbakevendende (recurrent) nevralt nettverk (RNN) som trenes ved hjelp av dynamisk programmering som tilnærmer optimal kontroll. Kontrollmetoden omtales som RNN-basert regulator. Den fjerde og siste regulatoren består av to feedforward nevrale nettverk som trenes ved hjelp av direkte heuristisk dynamisk programmering (dHDP).

Ytelsen til PI-ANN-, MPC-ANN- og dHDP-regulatorene er blitt evaluert og sammenlignet ved hjelp av simuleringsresultater hentet fra Matlab/Simulink-simuleringsmiljøet. Mens PI-ANN-kontrolleren ikke viser noen forbedring sammenlignet med den konvensjonelle PI-kontrolleren, yter MPC-ANN og dHDP betydelig bedre, og oppnår mindre oscillasjoner, forbedret dynamisk respons og robusthet mot parameterendringer. Ytelsen til PI-ANN og MPC-ANN-regulatorene har blitt verifisert ved å sammenligne simuleringsresultatene fra Matlab/Simulink med resultater fra OPAL-RT.

Table of Contents

List of Figures

List of Tables

Abbreviations

ADDHP action dependent DHP. 21

ADHDP action dependent HDP. 21

ADP adaptive dynamic programming. 21

AI artifical intelligence. 11

ANN artifical neural network. 13, 14, 20, 43, 48

CCS-MPC continuous control set MPC. 11, 36, 37, 39, 40, 42–48

DBC dead beat control. 11

DC direct current. 8, 10, 20, 25

dHDP direct HDP. 13, 22, 56, 60–65, 67–71

DHP dual heuristic programming. 21

DP dynamic programming. 49–51, 56

EMPC explicit model predictive control. 37, 38

FATT forward accumulation through time. 49, 51–54

FCS-MPC finite control set MPC. 11, 37, 38, 48

FFNN feedforward neural network. 14, 26, 28, 42, 49, 51, 59, 60

FL fuzzy logic. 11, 12

GN Gauss-Newton. 26, 27

GPC generalized predictive control. 37, 38

GTO gate turn-off thyristor. 5

HDP heuristic dynamic programming. 13, 21, 22, 56

HIL hardware-in-the-loop. 23, 34, 35, 47, 65

IGBT insulated gate bipolar transistor. 5, 23

IGCT integrated gate commutated thyristor. 5

KVL Kirchoffs voltage law. 24

LM Levenberg-Marquardt. 26, 27, 42, 51, 54, 59

logsig logarithmic sigmoid. 13

LPF low-pass filter. 9

LQR linear quadratic regulator. 37

LTI linear time invariant. 10, 39

MDP Markov Decision Process. 17

MIMO multiple-input, multiple-output. 11, 36

ML machine learning. 12

MMC modular multilevel converter. 5

MOSFET metal-oxide-semiconductor field effect transistor. 5

MPC model predictive control. 11, 15, 36–39, 42–45

MPC-ANN MPC-trained artifical neural network. 36, 42–48, 67–70

MPP maximum power point. 20

MPPT maximum power point tracking. 20

MSE mean squared error. 15, 28–30

OSS optimal switching sequence. 38

OSV optimal switching vector. 38

PI proportional-integral. 9, 10, 22, 23, 25, 26, 28, 30–32, 34, 35, 39, 56, 67–70

PI-ANN PI-trained artificial neural network. 23, 26, 30–32, 34, 35, 42, 67–70

PLL phase-locked loop. 9, 10, 23

PR proportional resonance. 10

PV photovoltaic. 12, 20

PWM pulse-width modulation. 5, 8, 10, 14, 23, 56

ReLu rectifier linear unit. 15

RL reinforcement learning. 12, 13, 17, 19–21, 56, 60

RNN recurrent neural network. 14, 15, 49–54, 72

SD steepest-descent. 26, 27

SL supervised learning. 12, 13, 17, 60

SMC sliding-mode control. 11

SSE sum of squared errors. 16, 26

SVPWM space vector PWM. 8

tansig hyperbolic tangent sigmoid. 13

THD total harmonic distortion. 22

TSK Takagi-Sugeno-Kang. 12

UL unsupervised learning. 12

VSC voltage source converter. 5, 6, 8, 10–12, 23

VSI voltage source inverter. 20

1

Introduction

1.1 Motivational Background

In 2020, the share of renewables in global electricity generation was 29%, and the renewable generation was expected to expand by more than 8% in 2021. This represents the fastest yearly growth since the 1970s [1]. This transition of the global power grid corresponds to an increased number of grid-connected nonlinear devices, which can result in reduced power grid inertia and stability issues unless measures are taken [2].

The integration of renewable energy sources into the power grid requires grid-connected converters, the control of which must provide fast, robust and reliable operation. The use of proportional plus integral (PI) control with pulse-width modulation remains a popular control method due to its simplicity, strong adaptability and reliability [3, 4]. This control method is designed in the synchronously rotating (dq) reference frame, where one PI controller is independently applied to the d- and q-axis currents by use of appropriate decoupling terms [3, 5]. This coordinate transformation enables the successful application of linear PI controllers to DC components and results in a linear-time-invariant system. Although this highly simplifies the control, assumptions such as linearity and time-invariance limit the overall system performance. In response, a considerable effort has been made in recent years by the scientific and industrial communities to address the research challenge of improving VSC control [6]. As a consequence, there exist numerous control methods, which can be broadly classified as linear, non-linear, predictive and artificial intelligence (AI)-based approaches.

Unlike the linear control methods (e.g. decoupled PI control) the nonlinear strategies exploit the fact that the averaged VSC model is non-linear by directly compensating for the nonlinear dynamics. These methods have shown improved dynamic performance, compared to the linear PI controller. However, unless advancements are implemented, they produce a variable switching frequency, which results in disadvantages such as resonance problems and reduced efficiency [6, 7].

Among the predictive control methods, model predictive control (MPC) has recently evolved as a promising method. The MPC principle is to predict the process output at future time instants and compute an optimal control signal that minimizes a cost function while ensuring that system constraints are met. Advantages include improved dynamic performance, easier handling of multiple-input, multiple-output cases, and increased flexibility compared to conventional control methods [8, 9].

AI-based control methods have in recent years become a heated research topic for power electronic systems. The development of the number of publications on the application of AI-based technology within design, control and maintenance of power electronic systems is plotted in Figure 1.1. Here it can be seen that since the sudden increase in 2007, the number of AI publications has been continuously increasing [10]. Today, there exist numerous AI-based technologies which provide

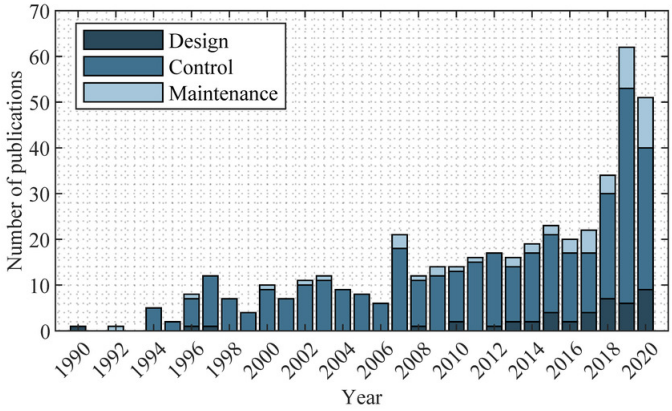


Figure 1.1: The development of AI-applications in power electronic systems, since 1990 (from [10]).

promising control methods for grid-connected converters. AI technologies aim to imitate human-like learning and reasoning [10]. One of the major AI techniques is machine learning. Approaches within machine learning can be classified as supervised learning (SL), unsupervised learning (UL) and reinforcement learning (RL). In SL the agent's goal is to learn the correct input-output mapping based on labeled data. This learning approach tries to emulate a behaviour defined by the user, and is relevant for control applications. UL, on the other hand, aims to discover a pattern in unlabeled data sets, and is primarily used for data clustering. RL, from a control aspect, is broadly related to SL, except that the agent is not fed labeled data. An RL agent learns by a reinforcement signal which either rewards or penalizes the agent's actions during interacting with the environment. By trial and error, the agent will eventually learn the correct input-output mappings, aiming to maximize the rewards.

Both the SL and RL approaches are employed for control applications. For both learning approaches, the use of artificial neural networks (ANNs) is increasingly drawing attention. In SL, common ANN-based control approaches consist of feeding the network with the inputs and outputs of an existing control method, such as linear, nonlinear or predictive controllers. In addition, through dynamic programming the network is fed the inputs and outputs of a user-defined optimal controller. Once the network is fully trained offline it is implemented as the controller. Within RL, one approach is to use the adaptive-critic ANN approach to develop approximately optimal ANN controllers [11]. In this approach the RL-based networks continue their training after implementation in the real system.

1.1.1 Problem Formulation

This thesis focuses on evaluating the performance of ANN-based control methods for the inner-loop current control of a grid-connected voltage source converter. Different control methods will be developed based on a literature review of prominent artificial neural network controllers. The controller performance will be analyzed using simulation results from the Matlab/Simulink environment and validated in the OPAL-RT simulation environment.

1.1.2 Related Work

The study of more robust, reliable and faster control for the grid-connected voltage source converter has been an important topic for several decades. Today, artificial intelligence is one of the most pertinent research areas, and much success has been achieved using artificial neural networks for grid-connected converter control.

In [6] an ANN of the feedforward type (FFNN) is trained on samples obtained by a model predictive controller (MPC) of the fixed control set (FCS) type. The resulting offline-trained network replaces the original FCS-MPC current controller in the system consisting of the voltage source converter

(VSC), a DC voltage source and a load. The simulation results show that the network controller has an improved performance compared to the original FCS-MPC. The total harmonic distortion (THD) is reduced and the transient response is faster and safer. A similar approach is used in [12] for a single-stage grid-connected photovoltaic (PV) system, but now for an MPC of the continuous control set (CCS)-type. The simulation results show that the CCS-MPC trained FFNN-based controller is able to efficiently extract the maximum power from the PV system while maintaining stable output signals under transient conditions.

In the aforementioned examples the network controller is trained using simulation data from an existing controller. In [13] a recurrent neural network (RNN) is employed as a current controller for the VSC. The network is offline-trained using a combination of the Levenberg-Marquandt (LM) algorithm and the forward accumulation through time (FATT) algorithm, to approximate optimal control. The simulation results and hardware experiments demonstrate better performance than conventional methods and show robust abilities concerning parameter changes. The RNN is also used in [14] to generate the reference current for the current controller of the VSC. The RNN is trained using an LM-based backpropagation algorithm, using the error signals between measured and reference current as input data. The training targets are new reference signals which remove the PI-generated steady state errors. The RNN-based controller simulation results are compared with conventional PI current control and show improved dynamic performance in terms of damping the oscillation introduced by the DC-link dynamics of the VSC.

There have also been developed network controllers which are further trained online after the initial offline training phase. In [15] a current controller consisting of two feedforward neural networks is trained using direct heuristic dynamic programming (dHDP). The controller is implemented in parallel to the conventional decoupled PI controller. Simulation results compare the performance to the original PI controller and show improvements in THD and transient response. The dHDP control method has also been applied to a power system stability control problem in [16], and for nonlinear tracking in [17, 18].

1.2 Scope and Objective

This thesis focuses on methods for improving the control of grid-connected converters using artificial intelligence techniques. The focus is limited to the inner-loop current control of the three-phase two-level voltage source converter. The objective of this thesis is to provide state-of-the-art ANN-based control methods for grid-connected converters, and evaluate and compare the control performances. The aim of this thesis can be summarized by two objectives:

1. Develop and implement different ANN-based control methods for the inner-loop current control for a grid-connected converter. Explore both the use of different network types and different network training methods.
2. By use of simulation results, evaluate and compare the performance of the chosen controllers for different control scenarios. Simulations will be performed using the Matlab/Simulink environment and validated using the OPAL RT environment.

Other elements which could improve the performance of the converter such as grid-synchronization techniques, filter design and digital signal processing, are outside the scope of this thesis.

1.3 Outline

Chapter 1 describes the motivational background of this thesis, which leads to the problem formulation. Related work is briefly described outlining relevant ANN-based control methods. Finally, the thesis' scope and objectives are stated.

Chapter 2 presents the theoretical background and is separated into four sections. First, important principles of the operation and control of the VSC are described. Next, an overview of relevant

AI technologies for converter control is made. The third section explains fundamental principles of artificial neural networks, including design- and training aspects. The fourth and final section gives a description of concepts and methods within reinforcement learning.

Chapter 3 describes the design and implementation of a PI-trained artificial neural network controller, denoted PI-ANN controller. First, the development of the decoupled PI control is described. Next, the architecture and training of the neural network is developed and the implementation is described. The performance of the PI-ANN controller is evaluated and compared to the conventional decoupled PI controller using simulation results from the Matlab/Simulink environment. In addition, a validation of the control performance is done by comparing it to corresponding simulations in the OPAL RT environment. Finally, all aforementioned results during both the design phase and the performance evaluation are discussed and important remarks are given.

Chapter 4 describes the design and implementation of a model predictive control trained artificial neural network (MPC-ANN) controller. The chapter starts by introducing the MPC, including its operation principles and existing approaches. Secondly, the design of the continuous control set (CCS) MPC type is thoroughly described. The third section gives the design of the network architecture of the MPC-ANN controller. The fourth section presents simulation results from the Matlab/Simulink environment. Based on these results the performance of the MPC-ANN controller and the CCS-MPC controller is evaluated and compared. In the next section the performance is validated by comparing the Maltab/Simulink simulation results to OPAL RT simulations. This chapter's final section discusses all previous observations.

Chapter 5 describes the design and implementation of a dynamic programming trained recurrent neural network (RNN)-based controller. The chapter starts by describing dynamic programming principles. Next, the RNN architecture is described. In the final section of this chapter the network training algorithm is described.

Chapter 6 describes the design and implementation of a direct heuristic dynamic programming (dHDP)-based controller and is comprised of three sections. First, the control design including an explanation of the dHDP algorithm and network architecture is described. Second, the controller parameters are determined by evaluating the controller performance for different values. The next section presents the simulation results obtained from the Matlab/Simulink environment, and the dHDP-based controller performance is evaluated. The final section discusses all previous observations.

Chapter 7 compares the performance of the PI-ANN, MPC-ANN and the dHDP controllers based on the simulation results from the Matlab/Simulink environment. This evaluation is separated into three parts, which consider different performance aspects. The performance of the different controllers are compared first during a varying reference current, second during a short circuit fault at grid-side, and finally for modified system parameters. The final section of this chapter discusses all observations.

Chapter 8 summarizes the main observations and concludes this thesis' results. In addition, proposals for further work are given.

1.4 Publication

Parts of the results of this project has been submitted for publication in [1].

[1] E. G. Øxnevad, P. R. Bana and M. Amin, 'Performance evaluation of ann-based control of a grid-connected converter with different training datasets', submitted to IEEE ACCESS, 2022.

2

Theoretical Background

This chapter serves as a background for all control methods studied in the following chapters. First, the operation and control of the voltage source converter will be thoroughly described in Section 2.1, next an overview of relevant artificial intelligence-based control methods is given in Section 2.2. In Section 2.3 essential elements of artificial neural networks are described. Finally, in Section 2.4 fundamental reinforcement learning concepts are explained and examples of relevant reinforcement learning methods for converter control are given.

In conjunction with this thesis a preceding specialization project was conducted and a scientific paper was submitted for publication. The topics and research described in this thesis include elements of both works.

2.1 Operation and Control of the Voltage Source Converter

This section introduces fundamental principles for the operation and control of the voltage source converter. In addition, an overview of some of the most commonly used current control methods is given.

2.1.1 Topology

The voltage source converter (VSC) can be classified depending on the topology used. The topology is specified by the number of phases and levels. For medium voltage converters for three-phase transmission it is common to use either two or three levels, whereas for high voltage applications the modular multilevel converter (MMC) is the ultimate choice [19–21]. In Figure 2.1 three different topologies are illustrated. The number of phases are defined by the number of legs, while the number of levels are defined by the number of transistors on each leg. The transistor usually consists of an insulated gate bipolar transistor (IGBT) and an anti-parallel diode, as is shown here.

A key property of all VSCs is the ability to self-commutate. This property is achieved by using fully controllable semiconductor switches, i.e. transistors [22]. Fully controllable means that the switches can be turned both on and off, such as the IGBT, integrated gate commutated thyristor (IGCT), gate turn-off thyristor (GTO) and metal-oxide-semiconductor field effect transistor (MOSFET). Most often, the choice is made between MOSFETs and IGBTs. The IGBTs are suitable for medium to high power applications and a frequency in the range of 20 kHz. On the other hand, MOSFETs are used for lower power, below 20kW, and a higher frequency in the range 20-800kHZ [23]. In this thesis the IGBT is chosen as this device is suitable for numerous applications. In fact, together with the pulse-width modulation (PWM) technique it is today one of the most widely

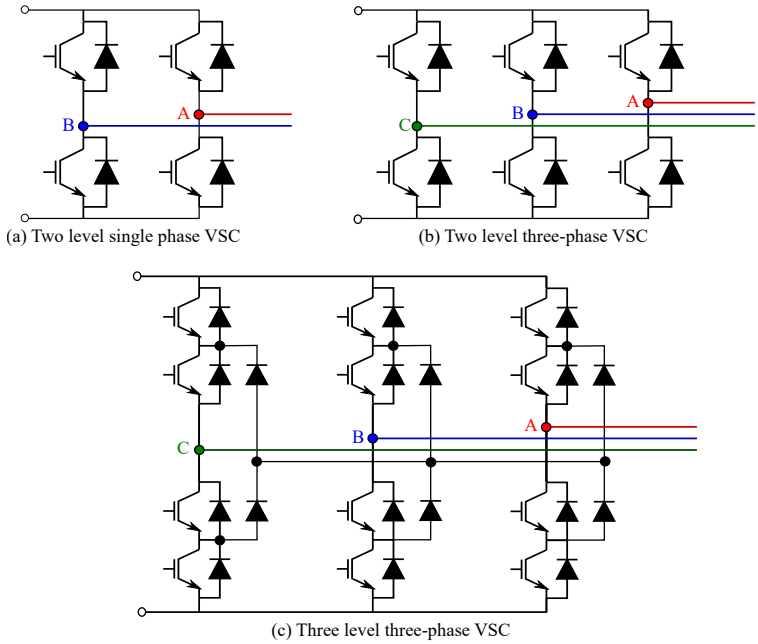


Figure 2.1: Examples of different voltage source converter topologies.

used configurations [24].

2.1.2 Operation

The operation of the VSC is best understood by investigating the output voltage v_c at each phase-leg independently. As illustrated for the two-level, three-phase VSC in Figure 2.1(b) each leg corresponds to each phase voltage, where the first leg provides the voltage pattern of phase a, the second leg corresponds to phase b and the third leg corresponds to phase c. Phase a of the converter output voltage $v_{c,a}$ at each switch position is shown in Figure 2.2. In (a) the upper switch is conducting and the voltage $v_{c,a}$ equals $\frac{1}{2}V_{DC}$. Figure (b) shows the commutation from switch to diode, here $v_{c,a}$ equals zero. In (c) the upper switch is turned off and the lower diode is conducting, this results in a voltage equal to $-\frac{1}{2}V_{DC}$. The reverse commutation from the lower diode to the upper switch is shown in (d). At each instant a maximum one switch is turned on. If both switches on one phase leg were simultaneously conducting this would lead to a short circuit with current values that could potentially damage the equipment. The above analysis considers the VSC operation in inverter mode, therefore at all times the upper diode and lower switch is turned off. For rectifier mode the upper switch and lower diode would remain off, while the upper diode and lower switch would be turned on and off subsequently. This bidirectional power flow is the reason why the VSC consists of anti-parallel diodes and switches.

2.1.3 Reference-Frame Theory

Reference-frame theory is today the basis for electric-drive simulation, analysis, and control design [25]. Reference-frame theory started when R. H. Park proposed a new method for analysing the synchronous machine [26]. The method consisted of replacing the voltages, currents and flux linkages, associated with the synchronous machine's stator windings, with variables associated with fictious windings rotating at the electrical angular velocity of the rotor. This led to a number of different transformations which in 1965 was proven to be contained in one general transformation which eliminates all position-varying inductances [27]. Today all control methods use one of three reference frames. The natural reference frame where all variables are in the original abc-coordinates, stationary reference frame where the variables are given in $\alpha\beta$-coordinates and the synchronous rotating reference frame where the variables are given in the dq-coordinates. Considering the

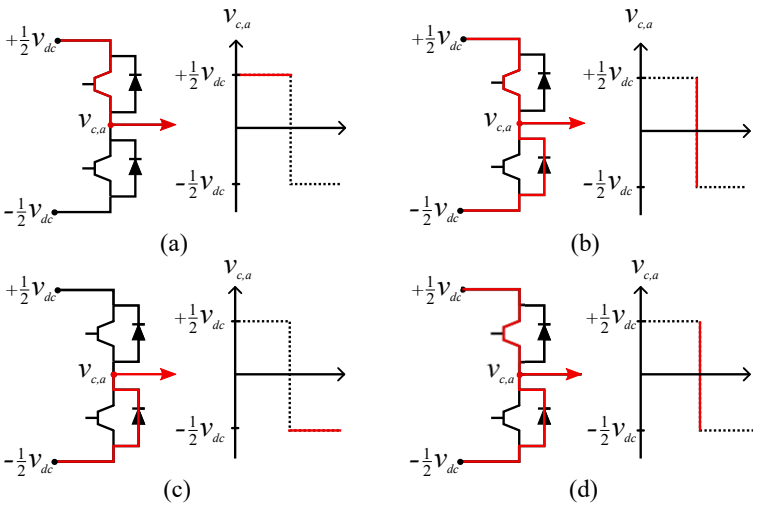


Figure 2.2: Basic operation of the voltage source converter.

general transformation, the stationary reference frame is obtained by referring the stator and the rotor variables to a frame of reference that rotates at zero angular velocity. The synchronous rotating reference frame, on the other hand, rotates at an angular velocity equal to the fundamental frequency of the grid.

Stationary Reference Frame

The stationary reference frame is obtained by transforming the three-phase voltages and currents into the two $\alpha\beta$-components, this is illustrated in Figure 2.3. This transformation is done using Clarke transformation which is defined by Equation 2.1. For a symmetric system $x_\gamma = 0$.

$$\begin{bmatrix} x_\alpha \\ x_\beta \\ x_\gamma \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos 0 & \cos(\frac{2\pi}{3}) & \cos(-\frac{2\pi}{3}) \\ \sin 0 & \sin(\frac{2\pi}{3}) & \sin(-\frac{2\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} \tag{2.1}$$

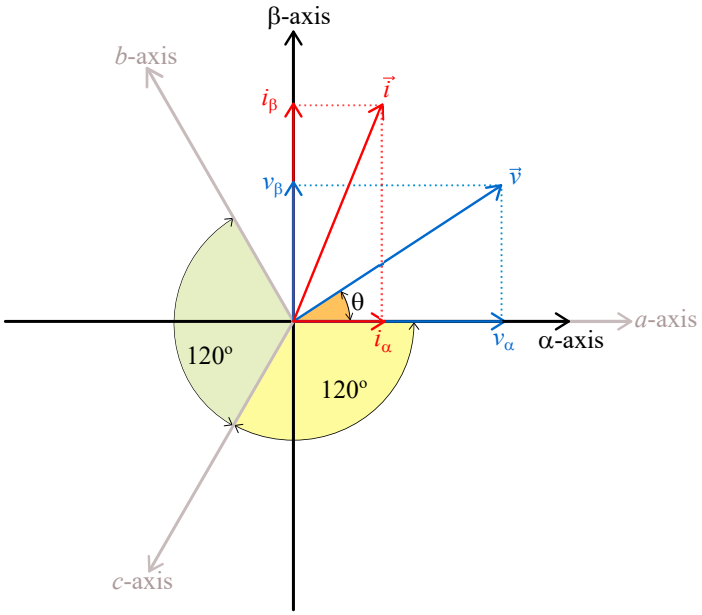


Figure 2.3: Stationary reference frame.

Synchronous Rotating Reference Frame

The synchronous rotating reference frame is obtained by transforming the $\alpha\beta$-components into dq-components. As illustrated in Figure 2.4 the reference frame is rotating synchronously with the grid voltage and the current and voltage now behave as direct current (DC)-components. The transformation from $\alpha\beta$ components to dq is done using Park transformation defined by Equation 2.2.

$$\begin{bmatrix} x_d \\ x_q \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} \tag{2.2}$$

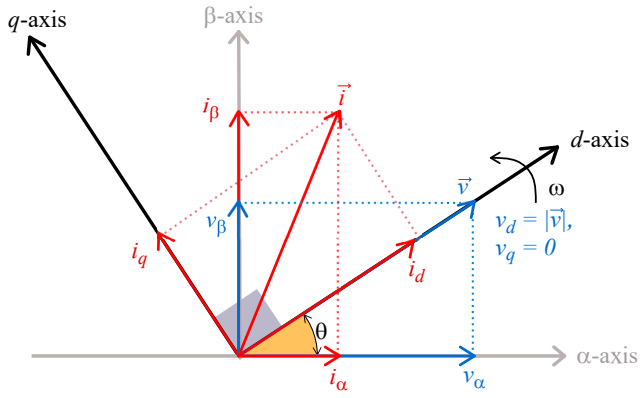


Figure 2.4: Rotating synchronous reference frame.

2.1.4 Pulse-Width Modulation

There are various PWM methods, where the two most widely used techniques for the VSC are the carrier-based sinusoidal PWM and the space vector PWM (SVPWM) [28], where linear control methods employ sinusoidal PWM and nonlinear control use SVPWM. There exist numerous discussions on the different PWM techniques, and for a more detailed description please refer to the following publications [29, 30].

Sinusoidal PWM

In sinusoidal PWM the the inner-loop current controller produces a control signal which is a sinusoidal modulation signal for each of the three phases. These are fed to the modulator as control signals with a frequency equal to the fundamental frequency. In the modulator these control signals are compared with a triangular carrier with a frequency equal to the desired switching frequency. This comparison leads to logical signals (high or low) which define the switching instant of each phase-leg. Since the two switches on each leg are operated in a complimentary manner, the switching state of all six switches $S_{A1}, ..., S_{C2}$ is easily determined. Sinusoidal PWM is illustrated for one of the three modulators in Figure 2.5, where the red signal is the control signal, i.e. output signal of inner-loop current controller, and the blue signal is the triangular carrier.

Space Vector PWM

SVPWM is another method for controlling the PWM of the converter. In this switching scheme, the voltages v_{abc} are first transformed to the stationary reference frame and are defined by Equation 2.3. Each of the three phase legs S_A, S_B and S_C consists of two switches, and can either be connected to the positive terminal or the negative terminal depending on each switching state. This results in two possible states for each phase leg. The phase leg S_A is equal to 1 if the upper

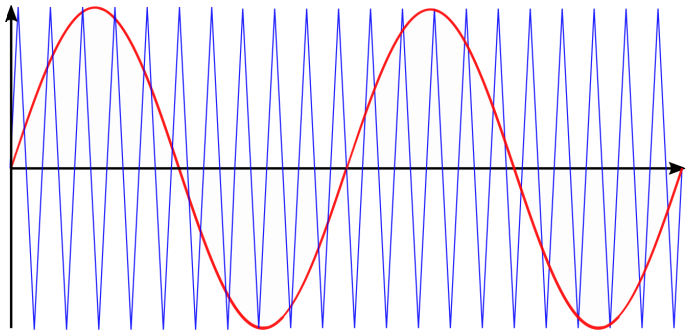


Figure 2.5: Sinusoidal PWM.

switch S_{A1} is turned on and S_{A2} is turned off. S_A is equal to zero and is connected to negative terminal if S_{A2} is turned on and S_{A1} turned off. Two switches on a phase leg can never be turned on simultaneously, which results in only these two possible states. Since the voltage vector in Equation 2.3 is a function of the states of all phase legs, the total number of possible voltage vectors is $2^3 = 8$, as presented by Figure 2.6. Here the voltage vectors v_0 and v_7 is the situation when all phase legs are connected to the negative terminal or the positive terminal, which results in a voltage equal to zero.

$$\begin{aligned} \mathbf{v}_{\alpha\beta} &= \frac{2}{3} V_{DC} \left(S_a + \alpha S_b + \alpha^2 S_c \right) \\ \alpha &= e^{j\frac{2\pi}{3}} \end{aligned} \tag{2.3}$$

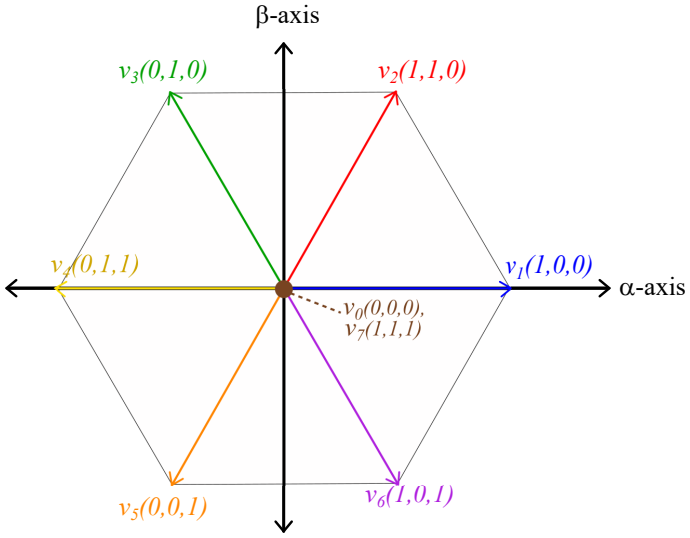


Figure 2.6: Space vectors.

2.1.5 Phase-Locked Loop

The phase-locked loop (PLL) is the most extensively employed grid synchronization technique [31] and is illustrated in Figure 2.7. The aim of the PLL is accurate tracking of the actual grid frequency f_g. Since the filtered converter output voltage v_f should be synchronized with the grid, these voltage measurements are implemented in the control loop. First the dq-axis voltages are obtained and passed through a low-pass filter (LPF). The actual phase angle error $\Delta\theta$ is obtained using the inverse tangent function on the filtered dq-axis voltages, $v_{PLL,dq}$. Since the voltage should be aligned with the d-axis the desired $\Delta\theta$ is zero, as is shown to the left of the summation. The proportional-integral (PI) controller takes $\Delta\theta$ as input and appropriately decreases or increases the angular speed ω_{PLL}. [24, 31]

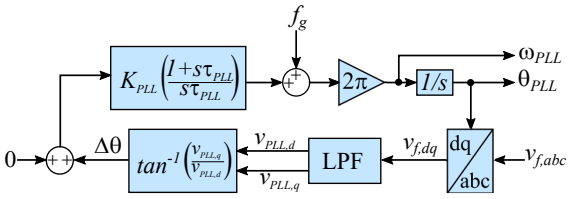


Figure 2.7: Phase Locked Loop.

There exist numerous grid synchronization techniques, where some of the most popular methods are the PLL-based technique, the zero-crossing method, filtering techniques and adaptive notch filter-based techniques [32]. In this thesis the PLL is chosen and testing of other techniques are considered outside the scope.

2.1.6 Cascaded Control

The standard control method for VSCs is to use cascaded control. An example of a cascaded control structure is presented in Figure 2.8. Here the converter control consists of an outer control loop which regulates the dc-voltage (v_{dc} and v_{dc}^*) and the reactive power (v_f and Q^*), and an inner control loop which regulates the converter current i_c. The task of the current controller is to regulate the current i_c such that it tracks the reference signal i_c^* obtained from the outer loop. This is done by comparing the reference and the actual measured signal, and based on the deviation generate appropriate modulation signals. Therefore the current controller is responsible for both error compensation and determination of appropriate switching [33]. The modulation signals m are then used to generate the switching pulses by use of PWM. The control strategy applied to grid-side converters usually consists of a fast inner current control loop which regulates the converter current and an outer loop which regulates the active power and the voltage/reactive power [34, 35].

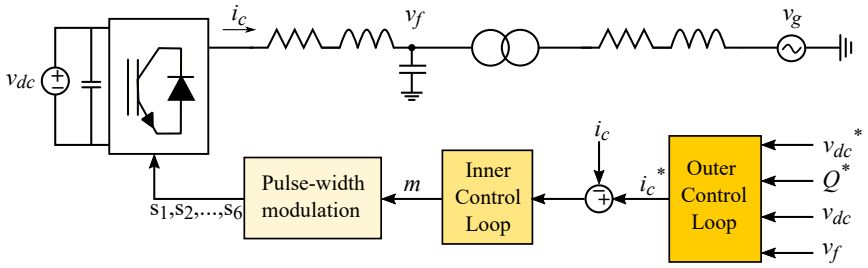


Figure 2.8: Cascaded control of the VSC.

2.1.7 Current Control Methods

Today, the control methods for the grid-connected converter can be broadly characterized as linear, nonlinear, predictive, and AI-based. The classification is shown in Figure 2.9, where in the lowest layer, some of the most common control methods are given. Among the linear control methods, an example is the aforementioned decoupled PI controller. This control method is designed in the synchronous rotating (dq) reference frame, which results in separate d- and q-axis error components which are regulated independently through a PI controller. This coordinate transformation enables a successful application of linear PI controllers to DC components and results in a linear time invariant (LTI) system with zero steady-state error. However, if a DC disturbance occurs in the three-phase system, steady-state errors will occur. As a remedy, the proportional resonance (PR) controller has been suggested [36]. This controller consists of a proportional term and a resonant term, and is designed in the stationary ($\alpha\beta$) frame. Compared to the PI controller, the PR offers a high tracking performance of the sinusoidal reference and rejection of disturbances [37].

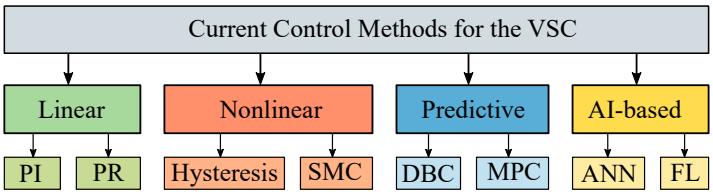


Figure 2.9: Classification of control methods for the current control of the VSC.

Unlike the linear controllers, the nonlinear control schemes exploit the nonlinear behaviour of the averaged VSC model by directly compensating for the nonlinear dynamics. Nonlinear control methods were first proposed in [38], and have been further developed in [39] and [40]. More robust nonlinear control has been achieved using sliding-mode control (SMC) [4, 41]. Although such nonlinear control strategies have been proven to improve the dynamic performance, unless advancements are implemented, they have the major disadvantage of producing a variable switching frequency which creates resonance problems and reduces the overall efficiency [6, 7].

Among the predictive control methods, dead beat control (DBC) was one of the earliest employed control methods [42, 43]. However in recent years, model predictive control (MPC) is receiving most attention and has proved to be a promising control method due to advantages such as improved dynamic performance, easier handling of multiple-input, multiple-output (MIMO) cases, and increased flexibility compared to conventional methods [8, 9]. The MPC principle is to predict the process output at future time instants and compute an optimal control signal which minimizes a cost function while ensuring that system constraints are met. The MPC strategies can be classified as continuous control set MPC (CCS-MPC) and finite control set MPC (FCS-MPC). The operating principles of these two control methods are principally similar but differ in the design of the cost function and the optimization stages [44]. In FCS-MPC the discrete nature of the converter is exploited and the control signals are switching signals which are directly applied to the converter. In CCS-MPC the cost function and the control signals are continuous, and the control signals are the modulation signals which are fed to a modulator. An MPC-based current controller will be developed in Chapter 4, please refer to this chapter for more information.

Artifical intelligence (AI) is currently one of the most pertinent research areas. Common for all AI technologies is the aim of producing intelligent systems which exhibit human-like learning and reasoning [10]. More information about AI-based control methods will be given in the proceeding section.

2.2 Artificial Intelligence Based Control for Converters

This section describes some of the most promising AI technologies for grid-connected converter control. The technologies are presented and classified in Figure 2.10.

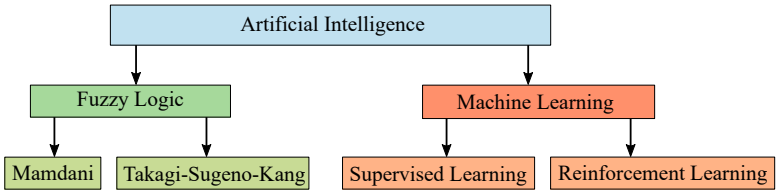


Figure 2.10: AI-technologies relevant for power electronic converter control.

2.2.1 Fuzzy Logic

As for all AI technologies, fuzzy logic (FL) tries to imitate human reasoning. It tries to model the imprecise modes of reasoning, which play an essential role in the ability to make rational decisions

in an environment of uncertainty and imprecision. Making such decisions depends on an ability to infer an approximate answer to a question based on stored, inexact and incomplete knowledge [45]. As an example, when determining a comfortable temperature level of the water before taking a shower, it is not simply a question of hot or cold, there are several temperature levels of comfort and discomfort in-between. Therefore, unlike Boolean logic which can only output two values; 1 for true, 0 for false, FL is multi-valued. The FL method consists of fours steps. First, the crisp input value is transformed into fuzzy values in a process called *fuzzification*, which is based on the information stored in the knowledge base. Next, the fuzzified inputs are aggregated with the fuzzy IF-THEN rules in the inference module, which imitates the human reasoning. Finally, the obtained fuzzy sets are transformed into crisp values in a process called *defuzzification*. This process is illustrated in Figure 2.11. FL-based control can be further separated into the Mamdani-type and the Takagi-Sugeno-Kang (TSK)-type. For further information, please refer to [46].

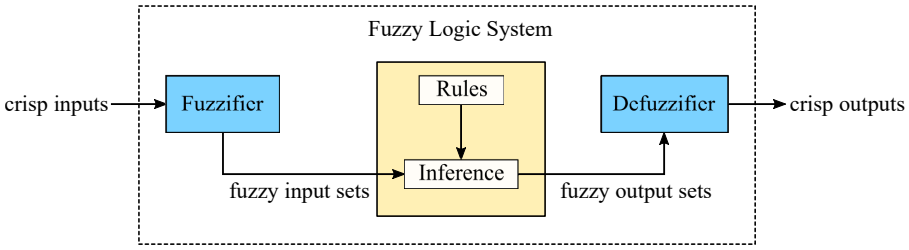


Figure 2.11: Fuzzy logic framework.

In [47] FL is applied to a photovoltaic (photovoltaic (PV)) sourced boost converter. In order to increase efficiency, the PV panels should be operated at the maximum power point, which varies according to the time of day and environmental changes. The FL controller was used to track the maximum power point, as it was able to rapidly respond to varying environmental conditions and being robust to changes of circuit parameters.

In [48] an FL controller is employed for islanded operation of an electronically interfaced distributed generation unit and its load. The controller regulates the voltage of the islanded system under varying load and uncertainties. The major result was that the FL controller is able to efficiently deal with the nonlinear system. The FL control design does not depend on the mathematical model of the system, and the controller was less sensitive to parameter variations compared to conventional controllers.

In [49] an FL-based current controller for the grid-connected VSC is proposed. The FL controller regulates d-axis and q-axis current components and is designed based on its frequency response on a bode diagram. The paper concludes that good control system performance can be achieved using FL-based control.

Compared to conventional controllers, FL-based controllers are cheaper to develop and are able to manage a wider range of operating conditions [50]. In addition, they are robust to parameter changes and handle nonlinear systems. A disadvantage of the FL controller is that its adaptability is limited which is due to the rule-based approach, i.e. it has no internal updating mechanism. Another disadvantage is that the design of the FL requires expert knowledge, which makes this method less available [51, 52].

2.2.2 Machine Learning

The machine learning (ML) category consists of three broad classes, supervised learning (SL), unsupervised learning (UL) and reinforcement learning (RL). In supervised learning (SL) the goal of the agent is to learn the correct input-output mapping based on labeled data, this learning method is mainly used for function fitting problems. In UL the agent aims to discover a pattern in unlabeled data sets, this method is mainly used for data clustering. In reinforcement learning (RL) the agent learns by a reinforcement signal which either rewards or penalizes the actions taken in an environment. Aiming to maximize the reward, the agent will eventually learn the correct

input-output mappings.

The SL and RL approaches are particularly interesting for control applications. For both learning-approaches the use of artifical neural networks (ANNs) for control related problems is increasingly drawing attention. In SL the ANN-based control methods mainly consist of training the network to approximate either an existing controller, as has been done in [6, 12] and [53] or an optimal one as was done in [54]. Within RL, a common approach is to use the adaptive-critic ANN approach to develop optimal network controllers [11]. This approach is within the large family of heuristic dynamic programming (HDP) methods. HDP methods commonly use the *Bellman optimality principle* to train the network to map inputs to actions (this will be further described in Section 2.4.1). Recently, the direct HDP (dHDP) method has shown to be a promising control method and has been applied to a large power system stability control problem in [16] and for nonlinear tracking in [17, 18].

These ANN-based control methods will be further described in Section 2.3, where the background for designing and training the ANN is given. In Section 2.4-based control methods will be described.

2.3 Artificial Neural Network

The ANN is inspired by the human brain function. Similar to the networks in our brain, ANNs can take different forms and sizes. However, most ANNs have a similar structure as presented in Figure 2.12, where the structure consists of an input layer, followed by hidden layers, which connect to the output layer. Each layer in an ANN is defined by the number of neurons and the connections between the layers are defined by activation functions. In Figure 2.13, a close up of the connection from the first input p_1 to the first neuron in the first layer a_1^1 is illustrated. The inputs to each layer come with a weight ($w_{i,j}$) and is summed together with a bias b, the result is a net input n which is fed to the activation function f. There are several activation functions, where the logarithmic sigmoid (logsig) function and the hyperbolic tangent sigmoid (tansig) function given by Equations 2.4 and 2.5, respectively, are common choices for the connection from input to hidden layer and for the connections within the hidden layer. For the connection from the hidden layer to the output layer a linear function is the common choice, since this enables all input-output mappings.

$$logsig(n) = \frac{1}{1+e^{-n}} \tag{2.4}$$

$$tansig(n) = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}, \tag{2.5}$$

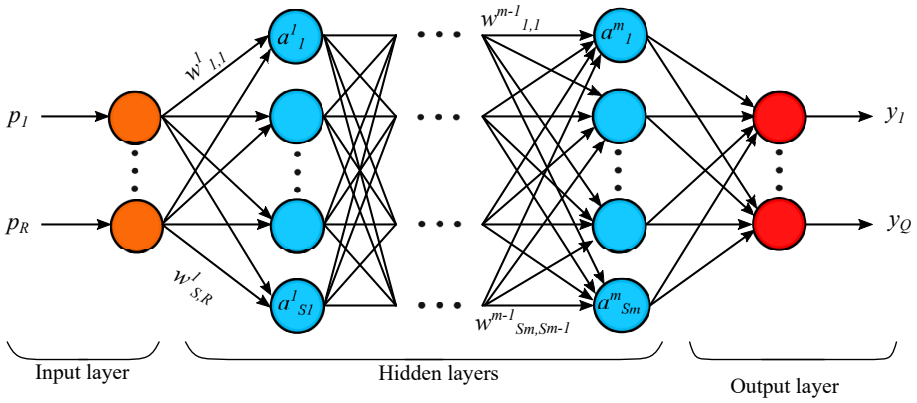


Figure 2.12: Fully-connected multi-layer feedforward neural network.

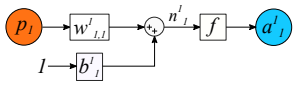


Figure 2.13: Single neural network connection, from the first input neuron p_1 to the first neuron in the hidden layer a_1^1.

2.3.1 Types of Artificial Neural Networks

There are two main types of the neural networks, the feeedforward neural network feedforward neural network (FFNN) and the recurrent neural network recurrent neural network (RNN). In the feedforward type the information only flows in the forward direction. In Figure 2.14 an RNN is presented, here the information can also flow backwards, illustrated by the dotted arrows, resulting in a cyclic flow of information. A network may be defined as either static or dynamic depending on the system it emulates. A system is considered static when the input-output relation is static, and dynamic if the relations are dynamic or temporal. The FFNN is used for static systems while the RNN is important for dynamic systems. The FFNN structure is the most widely applied in power electronics and motor drives. In fact in [46] it is stated that 90% of ANN applications use the feedforward type.

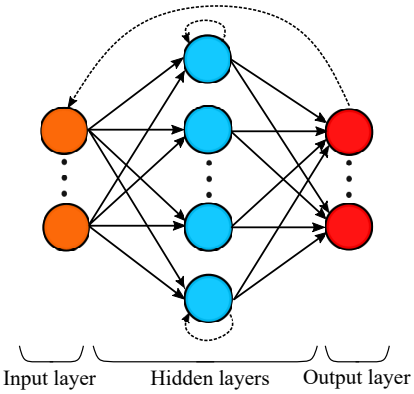


Figure 2.14: Example of a recurrent neural network.

2.3.2 Determining Network Architecture

Determining the network architecture consists of choosing an appropriate ANN type, sufficient number of inputs, outputs, the size of the hidden layers and the transfer functions between the layers.

The first step of designing the ANN is to define the problem the network is to solve. There are four major problem types; function fitting, pattern recognition, clustering and prediction. Since the network is to replace a controller, the problem can be said to be of either the fitting-type or the prediction type. Considering that the goal is to design a network which can perform as good as an existing controller or an approximated optimal controller the problem can be classified as function fitting.

A 2-layer FFNN was in [55] proven to be a universal function approximator, provided that the activation function in the hidden layer is of sigmoid-type, the output layer function is linear and that there are a sufficient amount of neurons in the hidden layer. Thus, this is one of the most applied networks for such problems. In fact, already in 1993 it was suggested to use FFNN for controlling a PWM boost converter [56]. Other FFNN-based control applications are found in the publications [6, 12, 57, 58]. However, the RNN is also a promising candidate being widely used to represent dynamic mapping of inputs and outputs. This is also possible using the FFNN with tapped delays, obtaining what is commonly known as the focused network, represented by Figure 2.15. To represent dynamic mapping this structure will however require a larger number of

neurons than the RNN [59]. The use of RNN-based control is described in publications [14, 60–62], to mention a few.

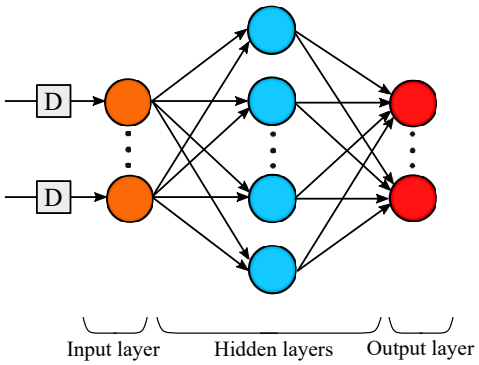


Figure 2.15: Focused neural network with time-delayed input, adapted from [63]

Regarding the activation functions these are chosen based on the previously mentioned recommendations regarding universal approximation. That is using one of the sigmoid functions in the hidden layer and a linear output at the output. However, it is interesting to note that different approaches have also been described in literature. In [57] the rectifier linear unit (ReLu) is used in the hidden layer. If the input to the ReLu function is positive, then the output will equal the input, if not the output is zero. At the output layer the Softmax function is chosen, this is a function which modifies each element of an input vector so that the sum of them equals 1. This way, each element can be interpreted as a probability. The choice of Softmax function at the output layer was also made in [64] and [65].

The number of inputs and outputs are determined by either the controller the network is to replace or the dynamic program used for generating training data. In [12] the network replaces and learns from the MPC, the number of inputs is therefore 7 ($i_{c,\alpha\beta}$, $i^*_{c,\alpha\beta}$, $v_{f,\alpha\beta}$ and v_{dc}) and the number of outputs is 3 (m_{abc}). Determining the number of hidden layers and the number of neurons within these layers is not as straightforward. A rule of thumb is that the more complex the problem is, the more layers and neurons are needed. However, as stated in [66] there is no theoretical reason to use more than two hidden layers, and one hidden layer is sufficient for most practical problems. Therefore a general approach is to start with one hidden layer and then increase the number of neurons and/or layers if the network performance is insufficient. The number of neurons in each hidden layer is usually determined by using trial and error.

2.3.3 Neural Network Training

The training of neural networks consists of updating the network parameters, i.e. the network weights and biases, such that the outputs of the network become as close as possible to the targets. The parameter vector is given by Equation 2.6, where each parameter x_i represents either an element in a weight or in a bias. Here the notation of the weight $w^k_{i,j}$ refers to the single weight element between neuron j at layer k and the neuron i of the subsequent hidden layer, the total number of layers is M and the number of neurons in layer m is S^m, and n is the total number of elements in all weights and biases, which is defined in Equation 2.7. The choice of training algorithm is firstly the definition of the updating rule. The rule is essentially the definition of the performance index $C(\boldsymbol{x})$, which is the measurement of how close the network outputs are to the targets. A popular performance index is the mean squared error (MSE) between the output of the network and the target. All performance indices are small when performance is good and large when performance is bad. So finding the weights and biases such that performance is optimized, is equivalent to finding the optimal parameter vector $\mathbf{x}$ such that $C(\boldsymbol{x})$ is minimized.[63, p.12-22]

$$\boldsymbol{x}^\top = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \dots & w^1_{S^1,R} & b^1_1 & \dots & b^1_{S^1} & w^2_{1,1} & \dots & b^M_{S^M} \end{bmatrix} \quad (2.6)$$

$$n = S^1(R+1) + S^2(S^1+1) + \ldots + S^M(S^{M-1}+1) \tag{2.7}$$

Generalization

A generalized network is a network that neither underfits nor overfits. Underfitting is when too few neurons are used in the hidden layers, such that the network is not able to accurately map inputs and outputs. As a consequence the network will have a poor performance both during training and when new data is presented. On the other hand, when too many neurons are used in the hidden layers, overfitting may occur. Then the network has a good performance during training, but performs badly when new data is presented. The remedy to underfitting is to increase the number of neurons. Inversely, the number of neurons may be decreased if overfitting occurs. However, other measures may also be taken to avoid overfitting, namely regularization or early stopping. Common for both methods is to restrict the magnitude of the weights and depend on setting aside a portion of the training set. Both methods are well-described in [63, Chapter 13], and the following is based on this.

The Early Stopping method is based on the fact that as training progresses, more of the weights are used for each iteration. Thus the complexity of the network is effectively increased by increasing the number of iterations used for training. Therefore, by ending the training before the final iteration is performed the complexity can be reduced. To perform early stopping the data set must first be separated into three subsets, one for training, one for testing and one for validation. Since all sets must represent all possible situations in which the trained network is to operate, a common procedure is to randomly distribute the data points among them. Typically 70 % of the data is used for training, while 15% is used for validation and 15% for testing. During validation the current training error is compared to the previous error. If this error is increasing for a specified number of consecutive iterations the training ends, and the final weights and biases are determined by the iteration before the error started to increase.

There are several regularization techniques, but common for all is to add a penalty term to the performance index which should be the sum of squared errors (SSE). An example is given by Equation 2.8. Here the complexity is controlled by α, which is made larger to achieve a smoother network response. The different regularization techniques are identified by how the regularization parameter (penalty term) is defined. In fact, for early stopping the regularization parameter is defined by minimizing the squared error on the validation set. Another common technique is the Bayesian regularization in which the parameter is set automatically. It has been proven in [67] that the Bayesian regularisation technique is better than the early stopping to avoid overfitting. Although this is perhaps an insufficiently detailed explanation of regularization techniques, Early Stopping was chosen for this thesis.

$$C(\boldsymbol{x}) = \sum_{q=1}^{Q} (\boldsymbol{t}_q - \boldsymbol{a}_q)^\top (\boldsymbol{t}_q - \boldsymbol{a}_q) + \alpha \sum_{i=1}^{n} x_i^2 \tag{2.8}$$

The Early Stopping method was chosen for two reasons. First of all, it is significantly easier to implement. Secondly, overfitting is in general a concern when the amount of data is limited. In this project data availability did not impose a limitation, as there are numerous situations which may be simulated, and corresponding data points which can therefore be stored.

Stopping Criteria

Ideally the network training ends when the training error converges to zero. In practical problems this rarely happens, thus other stopping criteria are usually defined. Since it is difficult to find the appropriate error value for which training is stopped, normal criteria are when either the gradient of the performance index, i.e. the cost, $\frac{\partial C(\boldsymbol{x}}{\partial \boldsymbol{x}}$ or the performance index reduction $C(\boldsymbol{x_{old}}) - C(\boldsymbol{x_{new}})$ reaches some predefined value. For both criteria a very small value must be chosen to avoid too early stopping. However, for most problems, when using the Early Stopping method, the validation error will increase before any of the stopping criteria are reached. [63, chapter 22]

2.4 Reinforcement Learning

RL is to learn how to map situations to actions in order to maximize a numerical reward signal. The learner, commonly known as the agent, and in our case the controller, has no knowledge of correct actions, but discovers the best actions by testing them and evaluating the reward for doing so. This principle is illustrated in Figure 2.16. This makes RL different from SL which, as previously explained in Subsection 2.2.2, is about learning from a labeled training set, and thus represent the expert knowledge of an external supervisor. During training of such a SL-agent each data point describes the situation defined by the inputs together with the correct actions to take in such a situation, defined by the targets.

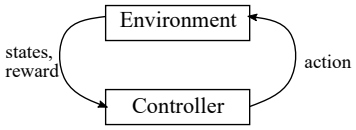


Figure 2.16: Principle of reinforcement learning.

The invent of RL was due to two large research topics, namely trial-and-error learning and optimal control. The early research of trial-and-error learning was nested in psychology as the study of behavioral psychology and became a known topic due to Edward Thorndike's 'Law of effects' [68]. The use of the trial-and-error learning in artificial intelligence dates back to the 1950s, where Farley and Clark designed a neural network to use trial-and-error-learning [69] and Minsky presented computational RL models [70]. Optimal control also began in the 1950s. In the beginning it was used to describe the problem of designing a controller to minimize a measure of a dynamical system's behavior over time [71]. In the late 80s thanks to Werbos in [72] and Watkins in [73], the merging of optimal control and trial-and-error learning resulted in what today is known as reinforcement learning.

2.4.1 Elements of Reinforcement Learning

Before proceeding it is necessary to introduce some core concepts within RL. This introduction gives a brief description and for an in-depth explanation please refer to [71].

Markov Decision Process

The Markov Decision Process (MDP) comes from dynamical systems theory and in RL is used to formulate the control problem as an optimal control of an incompletely-known MDP. The idea is to capture the fundamental aspects of the problem the agent is facing. The agent must have knowledge of the state of the environment, and it must be able act in such a way that the sensed states are affected. In addition the agent must have a goal which is related to these states. These three aspects, sensing of states, action with impacts and goal-orientation, are all included in MDPs. [71]

An MDP is normally described as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathtt{T}, R, \gamma \rangle$. Where $\mathcal{S}$ is a set of all states s, $\mathcal{A}$ is the set of all actions a. $\mathtt{T}$ is a transition model, $\mathtt{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Normally it specifies a conditional probability distribution, $\mathtt{T}(s, a, s') = \mathtt{Pr}(s_t = s | a_{t-1} = s, a_{t-1} = a)$. R is a reward function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Where $R(s, a)$ specifies how desirable it is to be in state s and take action a. For continuous task the reward function is given as

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

The expected value of the next reward is given by $\mathcal{R}(s, a, s') = \mathbb{E}\left[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right]$ γ is a discount factor, $0 < \gamma \leq 1$.[74]

Policy

The policy describes the behaviour of the agent for a specific time, and is essentially a mapping from states s to actions a, $\pi : \mathcal{S} \to \mathcal{A}$. The policy π, is essentially a function which specifies what action to take in each state. Normally it is defined as a probability, where $\pi(s, a)$ is the probability of taking action a in state s under policy π. However, in some algorithms it is defined as a look-up table. [74]

Value Functions

Most reinforcement learning algorithms consists of estimating the value functions. The state-value function $v^\pi(s)$ estimate how good it is for the agent to be in a given state s under the policy π. This measurement is done by estimating the expected total reward and is defined by Equation 2.9. Where $\mathbb{E}_\pi$ denotes the expected value given that the agent follows policy π.

$$v^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right] \tag{2.9}$$

The second value function is the action-value function $q^\pi(s, a)$. This measures how the value of taking action a in state s when following policy π. This is defined in Equation 2.10.

$$q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = a, a_t = a \right] \tag{2.10}$$

Both value functions satisfy a recursive relationship which is commonly known as the Bellman equation. This is shown for the v^π-function in Equation 2.11 and describes the relationship between the value of a state and the values of possible replacing states.

$$v^\pi(s) = \sum_a \pi(s, a) \sum_{s'} T(s, a, s') \left(\mathcal{R}(s, a, s') + \gamma v^\pi(s') \right) \tag{2.11}$$

The Bellman equation sets the basis for the *Bellman optimality equation* which states that "the value of a state s under an optimal policy π must equal the expected return for the best action from that state"[71]. This equation is based on the optimal state-value function v^* and the optimal action-value function q^*, which are defined by Equation 2.12.

$$v^*(s) = \max_\pi v^\pi(s), \quad q^*(s) = \max_\pi q^\pi(s, a) \tag{2.12}$$

The optimal action value function gives the expected return for taking action a in state s under the optimal policy π, thus it can be defined in terms of v^*. Vice versa, finding the maximum value of q^{π^*} over all actions $a \in \mathcal{A}$ results in the optimal state-value function v^*. Using these relations the Bellman optimality function for the v^*-function and q^*-function can be defined by Equations 2.13 and 2.14.

$$\begin{aligned} v^*(s) &= \max_{a \in \mathcal{A}(s)} q^{\pi^*}(s, a) \\ &= \max_a \mathbb{E} \left[r_{t+1} + \gamma v^*(s_{t+1}) | s_t = s, a_t = a \right] \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \left[\mathcal{R}(s, a, s') + \gamma v^*(s') \right] \end{aligned} \tag{2.13}$$

$$\begin{aligned} q^*(s) &= \mathbb{E} \left[r_{t+1} + \gamma v^*(s_{t+1}) | s_t = s, a_t = a \right] \\ &= \mathbb{E} \left[r_{t+1} + \gamma \max_{a'} q^*(s_{t+1}, a') | s_t = s, a_t = a \right] \\ &= \sum_{s'} T(s, a, s') \left[\mathcal{R}(s, a, s') + \gamma \max_{a'} q^*(s', a') \right] \end{aligned} \tag{2.14}$$

Finding the explicit solution of the Bellman optimality equation is rarely done in RL algorithms. Instead, many methods approximate the Bellman optimality equation, where actual measurements replace the expected transitions [71]. This will be described in more detail in Subsection 2.4.3.

Curse of Dimensionality

Normally, in control problems both states s and actions a are continuous. Thus, calculating the optimal action-value function and the state-value function values becomes very difficult, as the number of state-action pairs are approximately infinite [75]. In fact, the computational requirements grow exponentially with the number of state variables.

2.4.2 Classification of Reinforcement Learning Methods

Today there exist numerous RL-methods, which can broadly be separated into two categories, model-based and model-free. A third emerging category [76] integrates model-free and model-based methods, but is not considered in this thesis. Model-free methods are further separated based on the iteration procedure, which is either value-based or policy-based. The model-based methods are separated depending on whether the model is learned or already given. This classification is shown in Figure 2.17, where some of the most popular methods are stated.

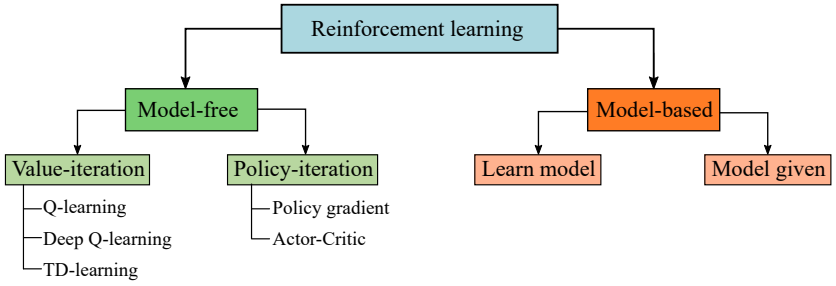


Figure 2.17: Classification of RL methods.

Model-Free Methods

Model-free methods learn by acting in the real environment, and the agent relies on trial-and-error experience for setting up the optimal policy. The two main approaches in this category are the value-iteration approach and the policy-iteration approach.

The value-iteration methods update the action-value function q based on the Bellman equation in 2.14 [76]. Such methods include Q-learning, deep Q-learning, temporal difference (TD) learning and many more. Q-learning is possibly the most popular RL-approach [69, 77]. According to [78], which reviews the application of RL in energy systems, half of the publications reported the use of Q-learning. In addition, many RL-algorithms use Q-learning as its underlying learning method [79].

The policy-iteration, also known as policy-based, methods parameterize the policy as $\pi(s, a|\theta)$. The target of such methods is to find an optimal θ either through gradient descent for an objective function $J(\pi)$ or by maximizing local approximations of J. This leads to the two methods policy gradient and actor-critic. [80]

Model-Based Methods

Whilst model-free methods inductively solve the problem, by using the agent's past experiences as evidence and statistics to estimate the value of its action, model-based methods deductively solve the problem, where the general understanding of the system is used by the agent to derive the most optimal action. Thus, in model-based methods the agent has knowledge of the model, this enables it to predict state transitions and future rewards. Provided that the model is correct this learning-method is much more sample-efficient compared to the model-free methods [80]. The model-based methods can be further separated into learn-model-method and given-model-methods. Although

there are several advantages of model-based methods, major drawbacks are difficulty of learning an appropriate model, and a worse asymptotic performance than the model-free approaches. The latter is due to assuming that the learned model accurately represents the actual environment, which results in model-bias [81]. Model-free approaches, on the other hand, can achieve better performance since they do not depend on the accuracy of the model, however they have the disadvantage of significantly higher sample complexity [82].

2.4.3 Relevant RL-Based Control Methods

Although there exist numerous RL-based control methods, there are few relevant examples. In general, most applications of RL-based control methods are used to improve the maximum power point tracking (MPPT) for renewable energy integration. The use of RL for MPPT is of course relevant, but translating this methodology for current control for the VSC is difficult. Similar observations are made in [78], where it is stated that "Most studies lack proper benchmarking" and about the RL's ability for problem solving it is stated that "only a limited number of publications has discussed its broad application".

Q-Learning Based Control

Q-learning is a model-free learning algorithm in which the environment is described by states $s \in \mathcal{S}$. The agent interacts with these states and chooses an action $a \in \mathcal{A}$ from which it receives a reward $r \in \mathcal{R}$. The goal is that the agent learns the mapping from the states to the long-term value of taking a particular action. This mapping is learned by means of a Q-function, which usually is the Bellman equation, as given by Equation 2.15, which is equivalent to Equation 2.14 with a slightly different notation. From this an optimal Q-value $Q^*_{s,a}$ is obtained and stored in a Q-table, indexed by the corresponding state and action pair required to obtain this Q-value. This table is updated every time an action is taken. [83]

$$Q^*(s,a) = E(r(s,a) + \gamma \max_{a'} Q^*(s',a') \tag{2.15}$$

In Q-learning the learning is done off-policy, meaning that the Q-function directly approximates the optimal Q-value, without considering the policy being followed. The policy only determines which state–action pairs are visited and updated [71].

In [84] a combination of ANN and Q-learning is used for MPPT of a wind energy conversion system. Here the MPPT algorithm is switched from the online Q-learning method to an optimal relation-based online MPPT algorithm when the maximum power point (MPP) is learned, and it is switched back again if the relation starts to deviate from the optimal relation. The method is compared to the conventional P&O (Perturbe and Observe) MPPT. Simulations are done under varying wind conditions and system aging conditions, the results show that when the wind is changed the conventional method searches for the new MPP without considering previous experience, which may lead to searches in the wrong direction and the efficiency is decreased. The proposed method is able to learn by previous experience, which results in a higher MPPT efficiency, less deviation from optimal values and the produced energy is higher, compared to the P&O method. Also in [85] Q-learning is used to improve the MPPT of a wind energy conversion system. However, here the authors use only Q-Learning and not ANN. In [86] and [87] RL is used to improve the MPPT for PV sources.

Despite an extensive search for Q-learning applied to current control for the VSC, only one publication was found. The publication [88] was the only relevant Q-learning method that was found. Here a Q-learning-based control methodology is presented and applied for controlling buck and boost converters. Simulation results showed that line and load regulation was achieved with the proposed RL-controller. In addition, suggestions are given for enabling a more robust and versatile controller. The paper states that the methodology can be extended to DC-AC converters, such as the voltage source inverter (VSI). The proposed control strategy is given in Figure 2.18. Here the first step is to initialize the system with a stable control policy which involves one of two strategies. Either (1) implementing the optimal controller after all steps are completed for an initial open-loop

control input. Or (2) the optimal controller is implemented using stabilized initial values. The next step is to delay the progress such that a steady state is reached, the delay may be of several samples and is important if the open-loop control input is to be used. The third step is to collect measurements of the states i and v and the fourth step is to pre-process the data and find the optimal solution in an iterative manner. The fifth and final step is to implement this optimal controller by sending the control signal to the converter. Although it would be interesting to test this method for the VSC, time was insufficient and for further details please refer to the original publication in [88].

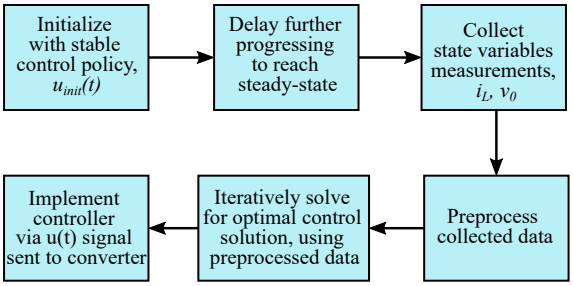


Figure 2.18: The proposed Q-learning-based control methodology (adapted from [88]).

Adaptive-Critic Based Control

Adaptive-critic methods are typically classified as approximate/adaptive dynamic programming (ADP), which are also commonly referred to as neurodynamic programming [89], and are defined as the family of methods that seek to find an (approximate) optimal control policy for a stochastic process whose model depends on unknown parameters [75, 90]. In these methods the curse of dimensionality problem is avoided by use of a system, called the critic, which approximates the dynamic programming cost function. The structure of the adaptive dynamic programming (ADP) approach is illustrated in Figure 2.19. Here the agent consists of two neural networks; the actor and the critic, which are both tuned online using the observed data. The critic approximates the cost function, taking into account the system states and reinforcement signal, which results in an updated policy. This policy update and the measured system states are then sent to the actor which computes and implements the appropriate control actions.

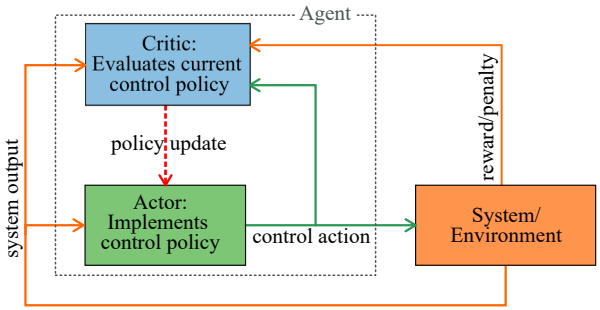


Figure 2.19: Structure of the actor-critic methods (adapted from [89, 91]).

According to [92] adaptive control algorithms was already a topic in the 50s, and became a popular research topic in the 1970s and 1980s. However the use of RL in such control methods is a fairly new approach. The importance of doing so, is that RL provides an adaptive control which converges to the optimal control. Thus, RL provides optimal adaptive controllers which learns online.

In [93] Werbos proposed two algorithms for implementing ADP, HDP and dual heuristic programming (DHP). Where HDP is a method for estimating the cost function and DHP is a method for estimating the gradient of the cost function. In addition, their corresponding action dependent (AD) versions, resulting in action dependent HDP (ADHDP) and action dependent DHP (ADDHP).

Today, among numerous ADP-based methods, HDP is one of the most basic and widely used structures [91]. The HDP principle is to estimate the cost function for a given policy. This estimation only requires the the instantaneous reinforcement signal $r(t)$ and is given in Equation 2.16. Note the equivalence to the previous value functions in Equations 2.9 and, The critic performs this estimation by minimizing the prediction error defined in Equation 2.17 over time.

$$\begin{aligned}\hat{J}(t) &= r(t) + \gamma \hat{J}(t+1) \\ &= \sum_{k=t}^{\infty} \gamma^{k-t} r(t)\end{aligned} \tag{2.16}$$

$$E_c = \frac{1}{2} \sum_t \left(\hat{J}(t) - U(t) - \gamma \hat{J}(t+1) \right)^2 \tag{2.17}$$

The actor use the knowledge of the states and the sum of the estimated cost and reinforcement signal to find optimal control action which are implemented to the model network. These steps are illustrated in Figure 2.20, where J is a cost function estimate, $x(t)$ are the measured states of the actual system, here referred to as the plant. u are the control inputs, γ is a discount factor and $r(t)$ is the reinforcement signal.

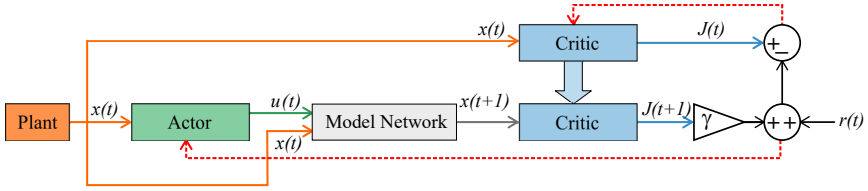


Figure 2.20: Principle of heuristic dynamic programming (adapted from [91]).

In [94] an HDP controller is used to control a boost converter. The paper states that the advantage of the HDP is enabling the boost converters to easily cope with large disturbances, and that the HDP with a well-trained critic and action networks can perform as an optimal controller for the boost converter. HDP is also used for optimal control in [95], where it is used for virtual inertia-based control of grid connected three-phase inverters. The paper shows that conventional controllers are not suited for non-inductive grids, but that the HDP controller enables the system to adjust itself to different conditions. Simulation results are included and confirm that the HDP controller outperforms the traditional direct-fed voltage and/or reactive power controllers in virtual inertia control schemes.

Recently, the direct HDP (dHDP) method has shown promising results. The dHDP estimates the controller parameters directly, unlike indirect versions which estimate model parameters before computing the control signals [89, 96]. The direct version avoids estimating the process, and enables the use of a model-free controller which is robust with respect to model uncertainties. The principle can be illustrated in Figure 2.20 by removing the model network.

In [16] the dHDP control method has been applied to a large power system stability control problem. Here the stability control problem consists of three main challenges; nonlinearities, uncertainties, and coordinate design. The paper proves that the direct HDP method can be employed to damp low-frequency oscillations. The control approach uses real system responses instead of the exact system model, and thus avoids influences of modeling for nonlinearities and uncertainties. A similar approach is used in [15] where a dHDP controller is designed to improve transient response and harmonics, in a system consisting of shunt active filters with current controlled VSCs. The proposed dHDP controller works supplementary to the PI control and efficiently tracks the d- and q-axis reference currents. Simulation results under different nonlinear load conditions are included and prove that the proposed controller significantly reduces the total harmonic distortion (THD), in addition to outperforming the conventional PI control method during transients.

3

PI-Trained Artificial Neural Network Control

In this chapter a PI-trained artificial neural network (PI-ANN) will be developed and implemented as a current controller for a two-level three-phase grid connected voltage source converter. The PI-ANN control design is separated into two parts. First, in Section 3.1 the decoupled PI controller is developed, including mathematical equations, analytical model, and tuning method. Third, in Section 3.2, the network design is described. This includes the training algorithm and architecture of the neural network. In Section 3.3 the performance of the finalized PI-ANN controller is evaluated by considering simulations results during different control scenarios. The controller performance is validated through hardware-in-the-loop (HIL) simulations in Section 3.4. Finally, in Section 3.5 the simulation results from both the design phase and the performance evaluation are discussed, and important remarks are made.

3.1 Decoupled Proportional-Integral Control

This section provides an explanation of the decoupled PI control. The controller will be implemented using sinusoidal PWM and is based on the lectures in the specialization course ELK-21 [97]. In Subsection 3.1.1 the mathematical representation of the system dynamics, in addition to the transformation from natural (abc) reference frame to synchronous rotating (dq) reference frame is described. Subsection 3.1.2 gives a detailed description of the decoupling of the d- and q-axis, and in Subsection 3.1.3 the tuning technique is described.

3.1.1 System Model

The topology of the converter considered in this project is the 2-level 3-phase VSC, where each phase leg consists of two IGBTs and anti-parallel diode. This converter will from here on be referred to as simply VSC. In Figure 3.1 the full system containing the grid-connected VSC is presented. This system consists of three main components. First the converter which outputs the three-phase converter voltage $v_{c,abc}$ and three-phase converter current $i_{c,abc}$. Secondly, the LC-filter represented by inductance L_f and capacitance C_f, in addition to a filter resistance R_f. At the output of the LC-filter is the filtered output voltage $v_{f,abc}$ and current $i_{f,abc}$ which are fed to the grid through a transformer. The third component is the distribution grid which is represented by resistance R_g, inductance L_g and the grid voltage $v_{g,abc}$. The corresponding system parameters are given in Table 3.1. In this project the carrier-based sinusoidal PWM method is used and grid synchronization is ensured using the PLL-technique.

Table 3.1: Parameters for the grid-connected VSC system

Symbol	Description	Value
V_g	Grid voltage	280 V
V_{dc}	DC-link voltage	500 V
P	Rated power	10 kW
f_{sw}	Switching frequency	8000 Hz
L_f	Filter inductance	1.55 mH
R_f	Filter resistance	10 mΩ
C_f	Filter capacitance	75 µF
L_g	Grid inductance	0.266 mH
R_g	Grid resistance	11.9 mΩ
C_{dc}	DC-link capacitance	3 mF
T_s	Sampling time	1 µs

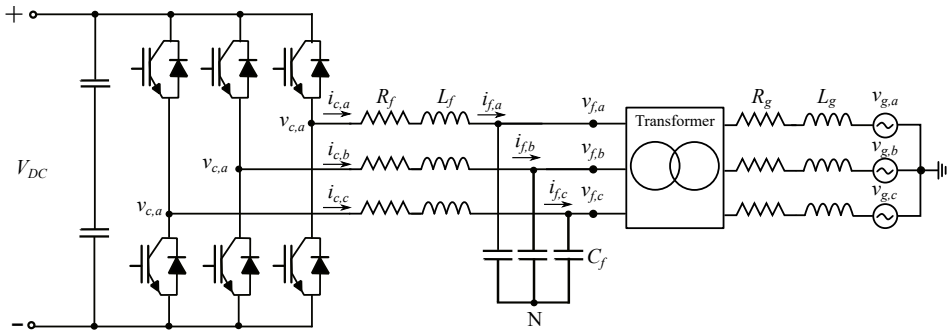


Figure 3.1: Topology of the grid-connected VSC.

By using Kirchoffs voltage law (KVL) on the grid side of the converter the mathematical representation of the system dynamics in Equation 3.1 is obtained.

$$\boldsymbol{v}_{c,abc} = R_f \boldsymbol{i}_{c,abc} + L_f \frac{d\boldsymbol{i}_{c,abc}}{dt} + \boldsymbol{v}_{f,abc} \tag{3.1}$$

Then by using the Park transformation matrix, previously given by Equation 2.2, dynamics in Equation 3.2 is obtained.

$$\boldsymbol{v}_{c,\alpha\beta} = R_f \boldsymbol{i}_{c,\alpha\beta} + L_f \frac{d\boldsymbol{i}_{c,\alpha\beta}}{dt} + \boldsymbol{v}_{f,\alpha\beta} \tag{3.2}$$

For transforming the components to the dq-frame it is useful to note that the relationship between the $\alpha\beta$-frame and the dq-frame is given by $\boldsymbol{x}_{\alpha\beta} = \boldsymbol{x}_{dq} e^{j\theta}$, where $\boldsymbol{x}$ represents either current or voltage. This results in Equation 3.3.

$$\boldsymbol{v}_{c,dq} e^{j\theta} = R_f \boldsymbol{i}_{c,dq} e^{j\theta} + L_f \frac{d}{dt} \boldsymbol{i}_{c,dq} e^{j\theta} + \boldsymbol{v}_{f,dq} e^{j\theta} \tag{3.3}$$

By using the product rule for $\frac{d}{dt}\boldsymbol{i}_{dq} e^{j\theta}$ and noting that the derivative of the phase angle $\frac{d\theta}{dt}$ equals ω, the term $e^{j\theta}$ is cancelled from both sides which results in Equation 3.4.

$$\boldsymbol{v}_{c,dq} = R_f \boldsymbol{i}_{c,dq} + j\omega L_f \boldsymbol{i}_{dq} + L_f \frac{d}{dt} \boldsymbol{i}_{c,dq} e^{j\theta} + \boldsymbol{v}_{f,dq} \tag{3.4}$$

By using that $ji_d = -i_q$ and $ji_q = i_d$ the system dynamics in dq-frame is obtained and given by Equation 3.5.

$$\begin{aligned} v_{c,d} &= R_f i_{c,d} - \omega L_f i_{c,q} + L_f \frac{di_{c,d}}{dt} + v_{f,d} \\ v_{c,q} &= R_f i_{c,q} + \omega L_f i_{c,d} + L_f \frac{di_{c,q}}{dt} + v_{f,q} \end{aligned} \tag{3.5}$$

3.1.2 Decoupling of D- and Q-axis

The decoupled PI controller is designed in the synchronously rotating (dq) reference frame and consists of two PI controllers. By use of decoupling terms each controller is independently applied to each of the d- and q-axis current deviations $\boldsymbol{\varepsilon}_{dq} = \boldsymbol{i}^*_{dq} - \boldsymbol{i}_{c,dq}$. This coordinate transformation enables the successful application of linear PI controllers to DC components and results in a linear-time-invariant system. [3, 5]

The development of the control starts with a Laplace transformation of the system dynamics previously given by Equation 3.5. Rearranging in terms of the converter current results in Equation 3.6.

$$\begin{aligned} i_{c,d} &= \frac{1}{sL_f + R_f}(v_{c,d} - v_{f,d} + \omega L_f i_{c,q}) \\ i_{c,q} &= \frac{1}{sL_f + R_f}(v_{c,q} - v_{f,q} - \omega L_f i_{c,d}) \end{aligned} \tag{3.6}$$

From this it is seen that the inductance terms $\omega L_f \boldsymbol{i}_{c,dq}$ and voltage terms $\boldsymbol{v}_{f,dq}$ must be fed forwards in order to compensate for undesirable cross-coupling effects, this results in the mathematical controller description in Equation 3.7.

$$\begin{aligned} v^*_{c,d} &= \left(K_p + \frac{K_i}{s}\right)(i^*_d - i_{c,d}) + v_{f,d} - \omega L_f i_{c,q} \\ v^*_{c,q} &= \left(K_p + \frac{K_i}{s}\right)(i^*_q - i_{c,q}) + v_{f,q} + \omega L_f i_{c,d} \end{aligned} \tag{3.7}$$

The obtained control signals $\boldsymbol{v}^*_{c,dq}$ must be transformed back to three-phase components $\boldsymbol{v}^*_{c,abc}$ using an inverse Park transform. The modulation indices $\boldsymbol{m}_{abc}$ are obtained after normalizing $\boldsymbol{v}^*_{c,abc}$ with the gain $\frac{2}{V_{dc}}$. This is illustrated in Figure 3.2.

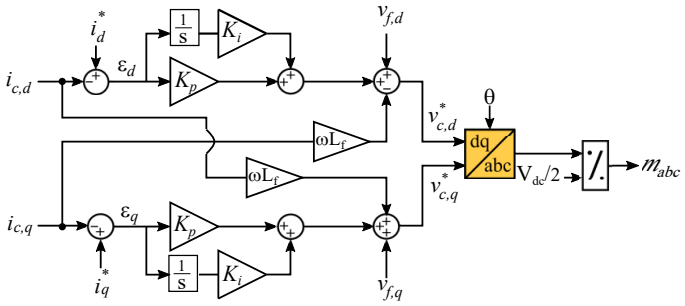


Figure 3.2: Decoupled proportional plus integral control.

3.1.3 Modulus Optimum Tuning Method

The proportional and integral gains K_p and K_i are determined using modulus optimum, a technique based on cancelling the dominant time constant, while keeping the closed loop gain larger than unity for as high frequencies as possible [98]. Modulus optimum achieves a fast response and is therefore a popular tuning method for the inner current loop. The tuning of each gain is summarized in Equation 3.8 for the per-unit value of filter inductance L_{pu} and resistance R_{pu}.

$$K_{p,pu} = \frac{\tau_{pu} R_{pu}}{2T_a}, \quad \tau_i = \tau_{pu} = \frac{L_{pu}}{\omega_b R_{pu}}, \quad K_i = \frac{K_p}{\tau_i} \tag{3.8}$$

For this system this tuning technique results in the gains

$$K_p = 4.133, \quad K_i = 25.8333$$

3.2 Network Design and Training

The structure of the PI-ANN control method is given by Figure 3.3. First the PI controller is used as an expert for generating the data needed for the offline training phase of the FFNN. The inputs and outputs of the PI controller are stored during simulations. Next, the network is trained using the simulation data. After completing the offline-training the network is implemented as a controller, replacing the original PI controller. However, before the network can replace the controller the training method and network structure must be determined. This will be described in the following subsections.

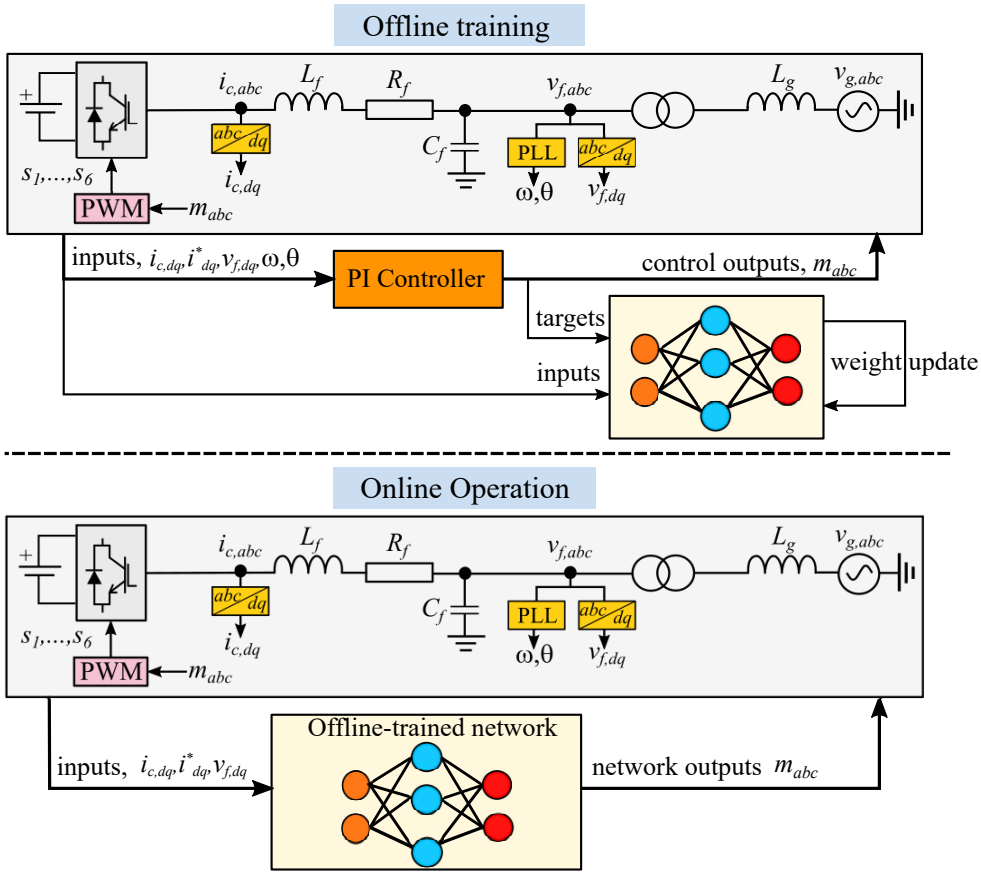


Figure 3.3: PI-ANN control design.

3.2.1 Levenberg-Marquardt Algorithm

In this report the Levenberg-Marquardt (LM) algorithm is chosen for training the FFNN. This algorithm was developed to solve nonlinear least square problems, and is based on a combination of the steepest-descent (SD) algorithm and the Gauss-Newton (GN) algorithm. The LM algorithm switches between these algorithms depending on whether the model is linear in its parameters $\boldsymbol{x}$ or not. This way when the parameters are distant from the optimal values, the problem becomes nonlinear and the SD method is employed. Conversely, when the parameters become close to their optimal values, the problem is close to quadratic, and the GN method is employed [63].

The LM algorithm measures the network performance during training as the SSE between the network outputs y and targets t_k at sample k. The resulting cost function $C(\boldsymbol{x})$ is given by Equation 3.9 for a total of Q samples, R inputs, M layers and S_i neurons in each layer i. The last summation is the sum over all elements in the error vector v_i, which is defined in Equation 3.10.

Here S_M is the number of neurons in the last layer, which gives a total of $N = Q \times S_M$ errors.

$$\begin{aligned} C(\boldsymbol{x}) &= \sum_{j=1}^{Q} (\boldsymbol{y} - \boldsymbol{t}_j)^\top (\boldsymbol{y} - \boldsymbol{t}_j) \\ &= \sum_{j=1}^{Q} \boldsymbol{e}_j^\top \boldsymbol{e}_j = \sum_{i=1}^{N} \boldsymbol{v}_i^2 \end{aligned} \tag{3.9}$$

$$\begin{aligned} \boldsymbol{v}^\top &= \begin{bmatrix} v_1 & v_2 & \dots & v_Q & v_{Q+1} & \dots & v_{Q \times S_M} \end{bmatrix} \\ &= \begin{bmatrix} e_{1,1} & e_{2,1} & \dots & e_{S_M,1} & e_{1,2} & \dots & e_{S_M,Q} \end{bmatrix} \end{aligned} \tag{3.10}$$

The aim of updating the weights and biases is to reduce the $C(\boldsymbol{x})$. This is seen by Equations 3.11 and 3.12, where the new weight $\mathbf{W}_{new}$ is the sum of the current weight $\mathbf{W}_{old}$ and the change in weight $\Delta \boldsymbol{W}$, where the change in weight is a function of the Jacobian $\mathbf{J}$. The Jacobian is defined in Equation 3.13 as the derivative of the error vector $\boldsymbol{v}$ with respect to the network parameters $\boldsymbol{x}$, and thus represents the change in error due to change in network parameters, N is the total number of errors and n is the total number of parameters as defined in Equation 2.7.

$$\mathbf{W}_{new} = \mathbf{W}_{old} + \Delta \mathbf{W} \tag{3.11}$$

$$\Delta \mathbf{W} = -(\mathbf{J}^\top \mathbf{J} + \mu \mathbf{I})^{-1} \mathbf{J}^\top \boldsymbol{v} \tag{3.12}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{3.13}$$

The switching between the SD method and the GN method is determined by the LM blending factor μ. If $\mu >> \mathbf{J}^\top \mathbf{J}$, then (3.12) approximates to $-\frac{1}{\mu} \mathbf{J}^\top \boldsymbol{v}$ which is the SD update. If μ, on the other hand, is small (3.12) becomes approximately $-(\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \boldsymbol{v}$, which is a GN update. In LM the cost function $C(\boldsymbol{x})$ is evaluated after each iteration, if it has not decreased, then μ is increased corresponding to a lower learning rate and a smaller step. If the performance has improved μ is decreased, and a larger step is taken.

The LM algorithm is illustrated in Figure 3.4. Here three stopping criteria are defined. First to stop the training after a fixed number of epochs, $Epoch^{max}$. Secondly the training is stopped if μ reaches a significant high value, μ^{max}. Third and last stopping criterion is to stop the training when a minimal value for the gradient of the cost function is reached, $\left[\frac{\partial C}{\partial \boldsymbol{w}}\right]^{min}$. Although the process is simple, the computation of specific elements is somewhat complicated. For more details of the computation details please refer to [63, Chapter 12].

In addition to the three aforementioned stopping criteria the Early Stopping method is used (Section 2.3.3), where the data is separated into three sets: 70% is used as training data, 15% is used for validation data and 15% is used for testing data. The number of validation errors is set to 6.

Initially, the SD algorithm was developed, this is slower than LM and was therefore not chosen for training the networks. However, the implementation is much simpler and is given in Appendix B. For the LM algorithm both embedded functions from the Matlab Deep Learning toolbox and an open access code from Github [99] were tested. The 'open access code' is slower than using the embedded functions from the 'Deep Learning toolbox', therefore the latter was used to produce the thesis' simulation results. In Appendix C the code for using the 'Deep Learning toolbox' is given.

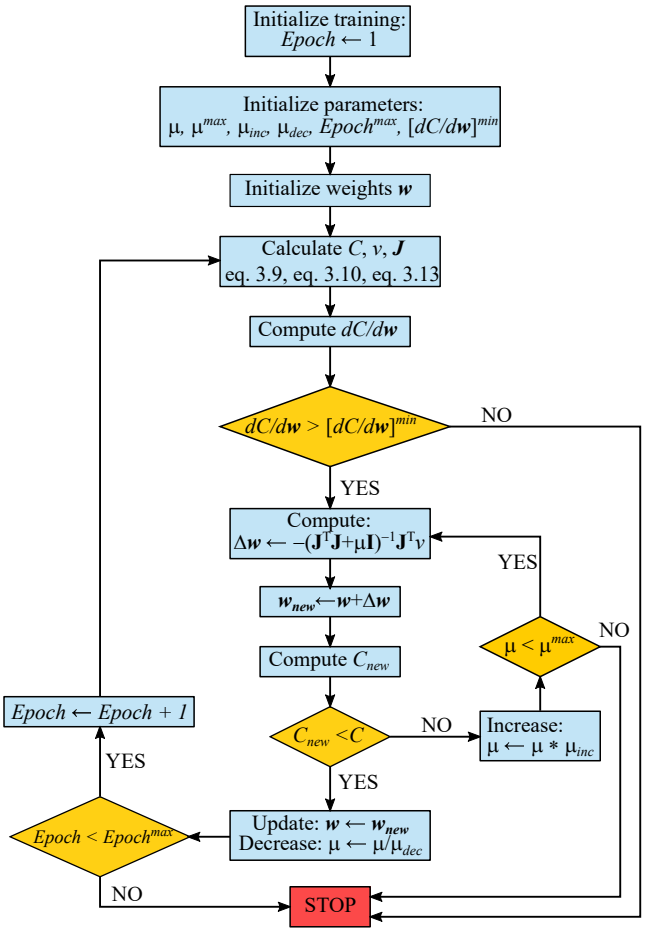


Figure 3.4: Levenberg-Marquardt algorithm.

3.2.2 Network Architecture

The design of the appropriate network architecture is concerned about choosing the appropriate number of neurons and layers in the network. As stated in Section 2.3.2 the number of inputs and outputs of the FFNN is the same as the number of inputs and outputs of the expert. However, there are several options when it comes to determining what is the actual input and output of the expert. E.g. one could choose the measurements of the currents and voltages as the inputs, i.e. the three-phase components $i_{c,abc}$ instead of the components i_{dq}. Since this corresponds to a more complex architecture this would be a bad choice. Instead, the four structures shown in Figure 3.5 were tested.

The first structure in Figure 3.5(a) fully replaces the decoupled PI controller, as it takes as input all the necessary components for performing the decoupled PI control, and outputs the optimal voltage vector in dq-frame. The second structure in (b) is almost identical to (a), except that the angular speed ω is removed. The structure in (c) has the same inputs as (a), but outputs the modulation indices m_{abc}, therefore the output of this network is sent directly to the modulator. The last structure in (d) takes the error signal ε_{dq} instead of the two current signals i_{dq} and i^*_{dq}, resulting in 5 inputs instead of 7. The output is the same as the previous structure, the modulation signals.

All structures were tested with a single hidden layer consisting of 20 neurons and the network was trained for a duration of 1000 epochs. The resulting network performance during training measured by the MSE is given by Figure 3.6(a). It can be seen that the performance is significantly poorer for structures 3 and 4, as the network does not manage to learn the correct input-output mappings, corresponding to relatively high values for the MSE. In addition, not shown in the figure, due to

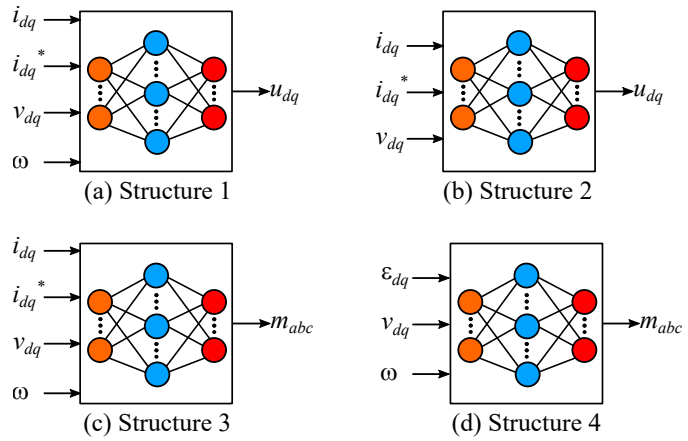


Figure 3.5: Overview of the different network input and outputs, resulting in four different network structures.

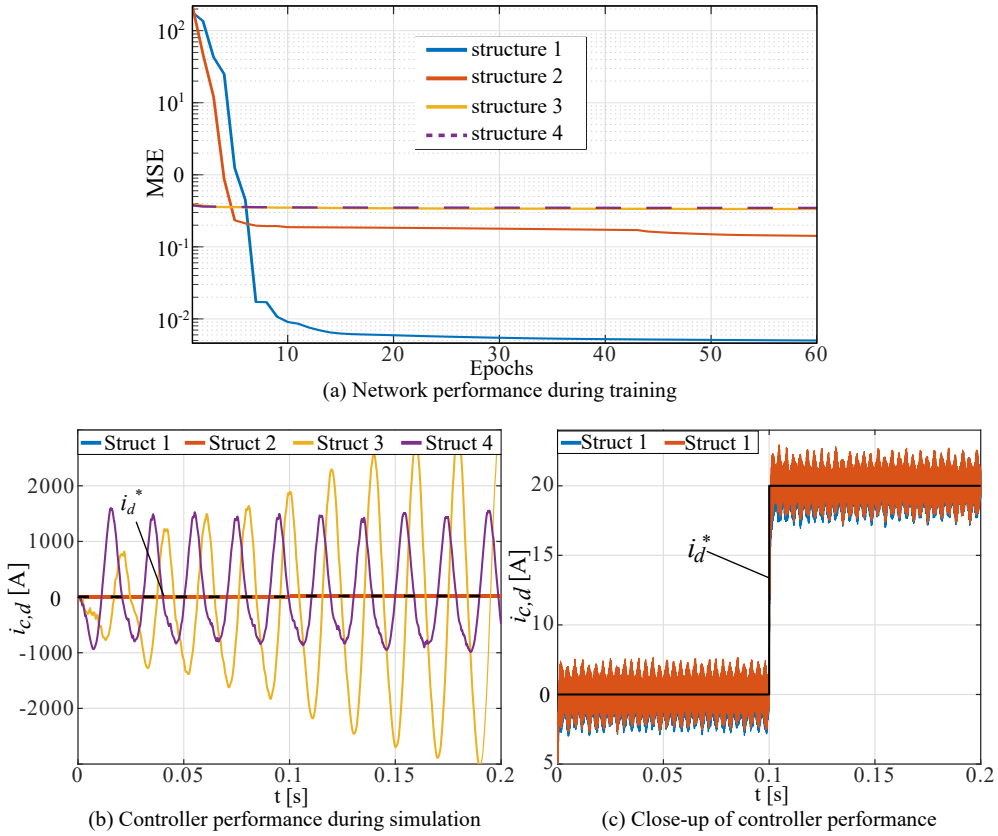
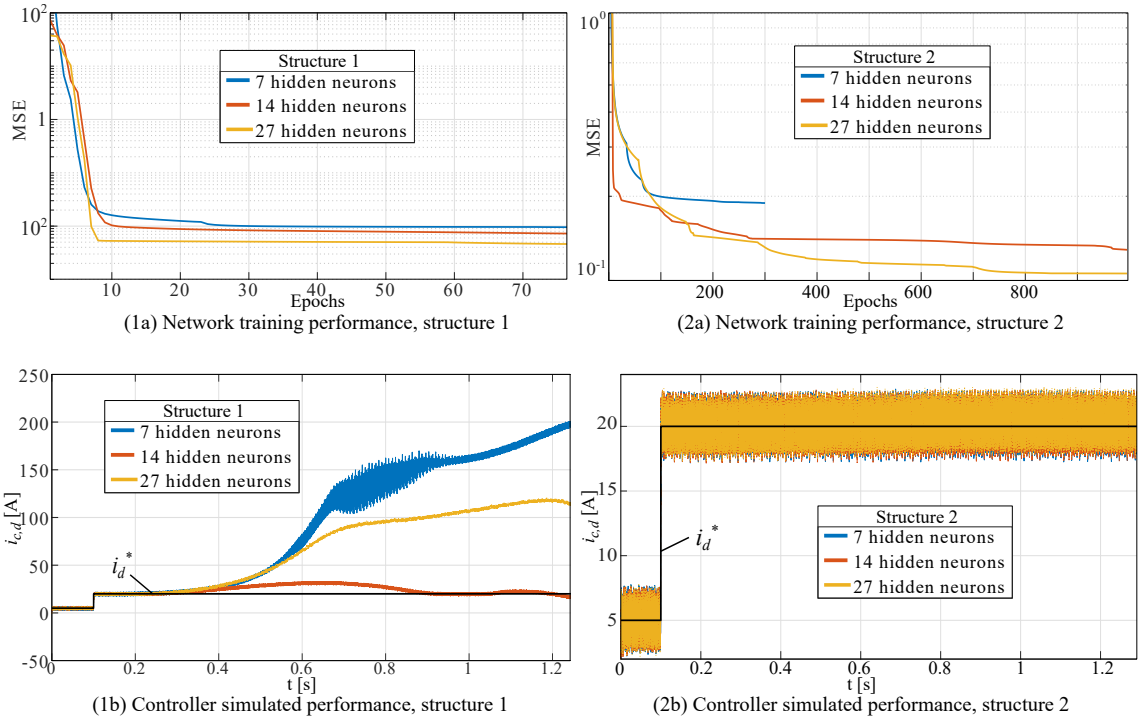


Figure 3.6: Network performance during training and tracking performance during simulation for all network structures.

validation errors the training stopped at 400 epochs for structure 3, and at 90 epochs for structure 4. The resulting controller performances using each network structure is plotted in Figure 3.6(b) and (c). Structures 3 and 4 both result in a controller which is unstable, and produces a converter current which neither follows the amplitude of the reference nor the phase. The close-up in Figure Figure 3.6(c) shows that although the MSE during training was significantly higher for structure 2 than structure 1, the controller performance when implemented is largely comparable.

The structures 1 and 2 were further analyzed for a longer simulation time and using different hidden layer sizes. The resulting training and controller performance of structure 1 using the network structures 7-7-2, 7-14-2 and 7-27-3 is plotted Figures 3.7(1a) and (1b), respectively. The different



(1a) Network training performance, structure 1

(2a) Network training performance, structure 2

(1b) Controller simulated performance, structure 1

(2b) Controller simulated performance, structure 2

Figure 3.7: Impact of hidden layer sizes on network performance during training (a) and controller performance during simulation (b). For structure 1 and structure 2.

network structures were trained using 100 epochs, because 1000 epochs resulted in overfitting, which gave an even poorer controller performance than the presented results. In Figure 3.7(1a) it is seen that the low MSE, values when using 7 and 14 hidden neurons, does not correspond to a good controller performance. For these networks the measured d-axis converter current $i_{c,d}$ drifts off even though the reference current i_d^* remains at 20 A. Since the mathematical equations that the network is trying to learn during training are simple linear relations, it is expected that overfitting is an issue. The same number of hidden neurons were also tested for structure 2, but the number of epochs was kept equal to 1000. The performance during training and simulation is plotted in Figures 3.7(2a) and (2b), respectively. Here it is seen that the controller performance is significantly improved as the measured current now remains at the reference value.

In Table 3.2 the total harmonic distortion of the filtered output current is given. Based on this structure 2 with 27 hidden neurons is chosen.

Table 3.2: Comparative analysis in terms of current THD [%].

Hidden neurons	Structure 2
7	4.07
14	4.05
27	3.85

3.3 Performance Evaluation

In this section, the performance of the PI-ANN controller is evaluated and compared to the conventional decoupled PI controller. This analysis considers three different test cases; varying reference current, short circuit fault at the distribution grid, and parameter uncertainty. Further details of each scenario will be given in the following subsections. The implementation of the decoupled PI controller is illustrated in Figure 3.2, and the implementation of the PI-ANN controller implement-

ation is illustrated in Figure 3.3. The simulations are performed using the controller parameters given in Table 3.3 and the system parameters given above in Table 3.1.

Table 3.3: PI and PI-ANN controller parameters.

Symbol	Description	Value
K_p	Proportional gain	4.133
K_i	Integral gain	25.833
	Network structure	6-27-2
$Epoch^{max}$	Maximum number of epochs	1000
μ_0	Initial blending factor	0.01
μ^{max}	Maximum blending factor	10^{10}
μ_{inc}, μ_{dec}	Increase/decrease of blending factor	10
$\frac{\partial C}{\partial x}^{min}$	Minimum gradient	10^{-7}
E_{val}	Maximum validation errors	6

3.3.1 Current Reference Tracking

A simulation is carried out for both the PI controller and the PI-ANN controller by varying the d-axis reference current magnitude (i_d^*) continuously and step-wise. The obtained results under this test case are presented in Figure 3.8. Here, the successful implementation of both controllers can be verified as the actual d-axis converter current ($i_{c,d}$) is tracking the corresponding reference component. However, deviations are seen, between the measured current and its reference, for both controllers. The tracking performance appears identical for both controllers. The waveforms of phase a of the filtered output current $i_{f,a}$ and output voltage $v_{f,a}$ are also included in Figure 3.8. For both controllers it is seen that the current waveform is affected by the reference current step variations, where momentary distortions are observed when the changes occur. The distortions decreases gradually, but neither controller manages to obtain a smooth current waveform before a new reference step is introduced. The voltage waveform appear unaffected by the reference current steps and remains acceptable smooth for the both controllers.

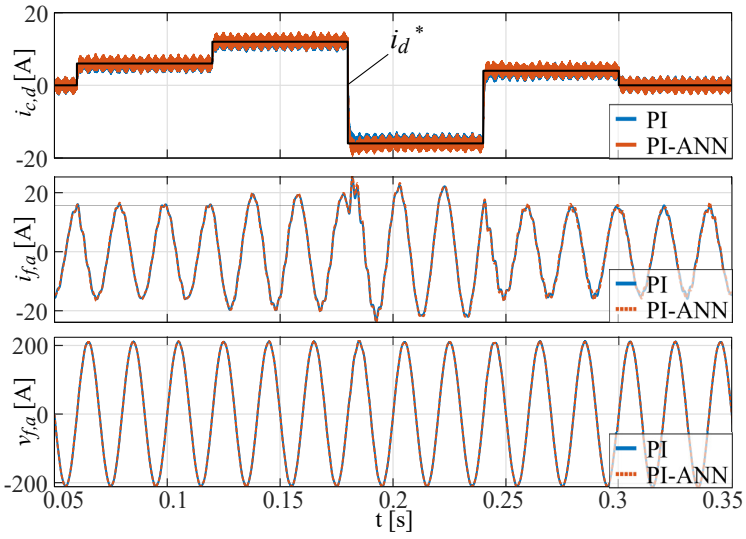


Figure 3.8: Performance evaluation of the PI-ANN controller and the PI controller under reference current variation.

3.3.2 Analysis During Grid-Side Fault

The performance of both controllers is also tested under grid-side fault conditions by reducing the grid voltage amplitude symmetrically to 20%, while keeping the reference current i_d^* magnitude constant at 18A. As presented in Figure 3.9, the fault is introduced at 0.2s and lasts for 3 cycles. In this context, the inability of the PI-ANN controller to tackle the fault can be observed from the converter current $i_{c,d}$ plot, which has large oscillations and obtains an unexpected high magnitude. In contrast, the PI controller operates at the expected level and produces a converter current with an expected magnitude and less oscillations. Similar observations are made for the current $i_{f,a}$ and voltage $v_{f,a}$ waveforms. For the PI-ANN the current has large oscillations at the introduction and clearing of the fault, in between these time instants the current amplitude is raised to an unexpectedly high value. The PI controller, on the other hand, produces a current with lower oscillations and an amplitude which is closer to the expected value. The voltage waveform is more similar for the two controllers, however the oscillations are larger for the PI-ANN controller at the introduction of the short circuit fault. During the fault the controllers produce a voltage with equal amplitude, but a small phase shift is observed between the two voltage waveforms.

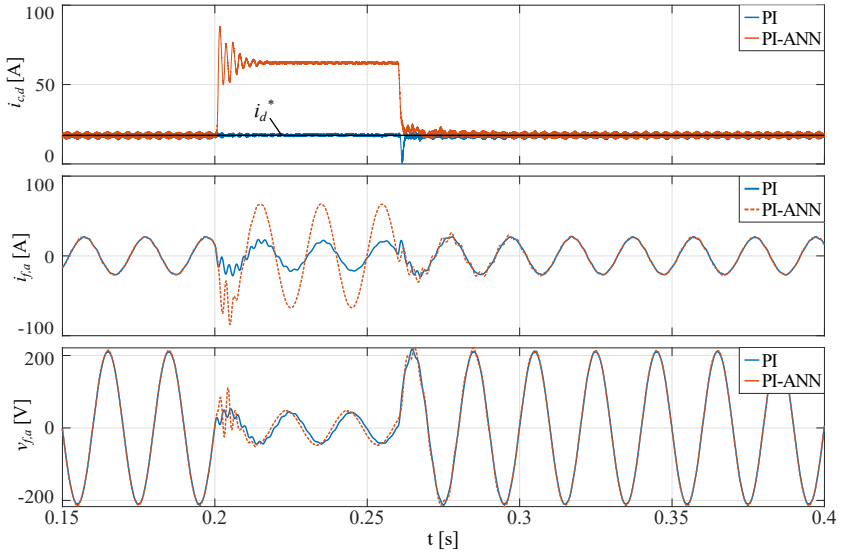


Figure 3.9: Simulation results of the PI-ANN controller and the PI controller under grid-side fault conditions.

3.3.3 Parameter Uncertainty

The robustness of the current controllers is further verified under the parameter uncertainty test case. This scenario considers the controller performance for modified grid and filter inductance. The grid inductance L_g is both decreased and increased by 300%, while the filter inductance L_f is increased and decreased by 20 %. Further, for easier evaluation of the controller tracking performance, the reference current (i_d^*) value is also changed to a different level at 0.2s.

In Figure 3.10 the $i_{c,d}$ and i_d^* are presented. It can be seen that the performance of both the PI controller and the PI-ANN controller is influenced by this change in parameters. This is observed by the deviation between measured and reference converter current, which is altered depending on the specific parameter change. For both controllers, the smallest deviation is observed for the increased L_f-value and the largest deviation is seen when the L_f-value is low. However, for the modification in grid inductance, the tracking performance appear the same. In all cases the $i_{c,d}$ waveform oscillates around its reference component for both controllers. However, both controllers manage to track the reference current for all parameter changes and the overall tracking performance is comparable for the two controllers.

In Figure 3.11 phase-a of the filtered output current is presented. For both controllers a smooth

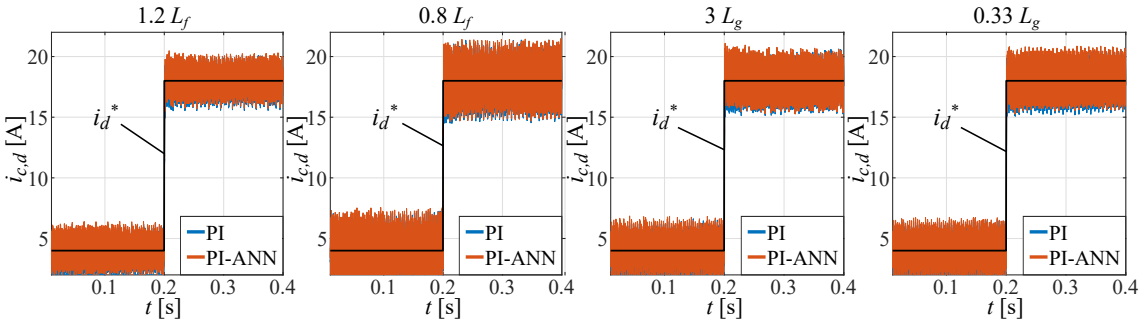


Figure 3.10: D-axis converter current under parameter uncertainty condition. For the PI and the PI-ANN controller

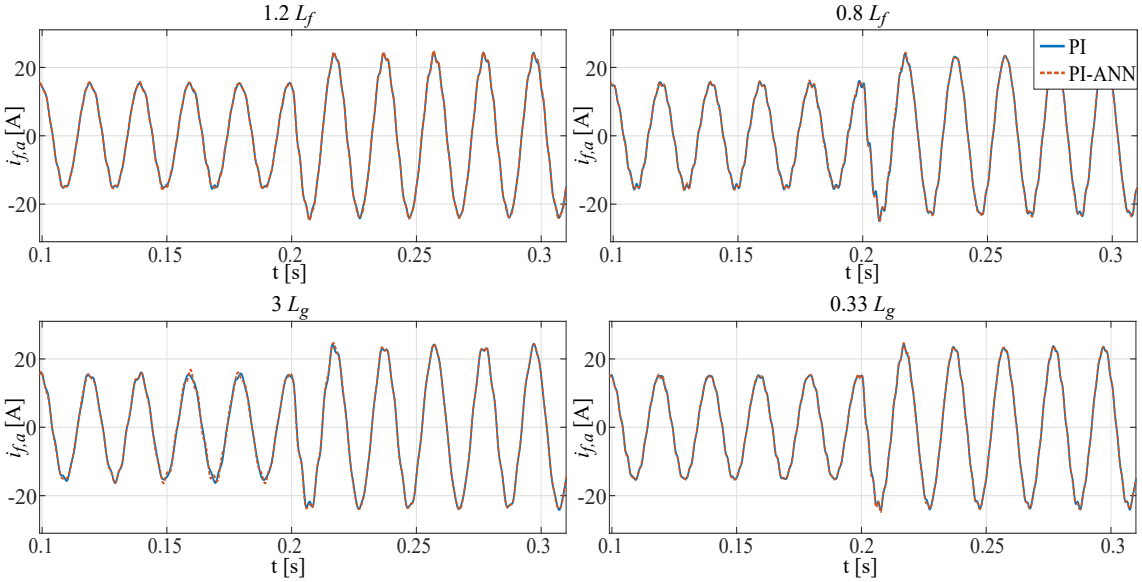


Figure 3.11: Filtered output current under the parameter uncertainty condition.

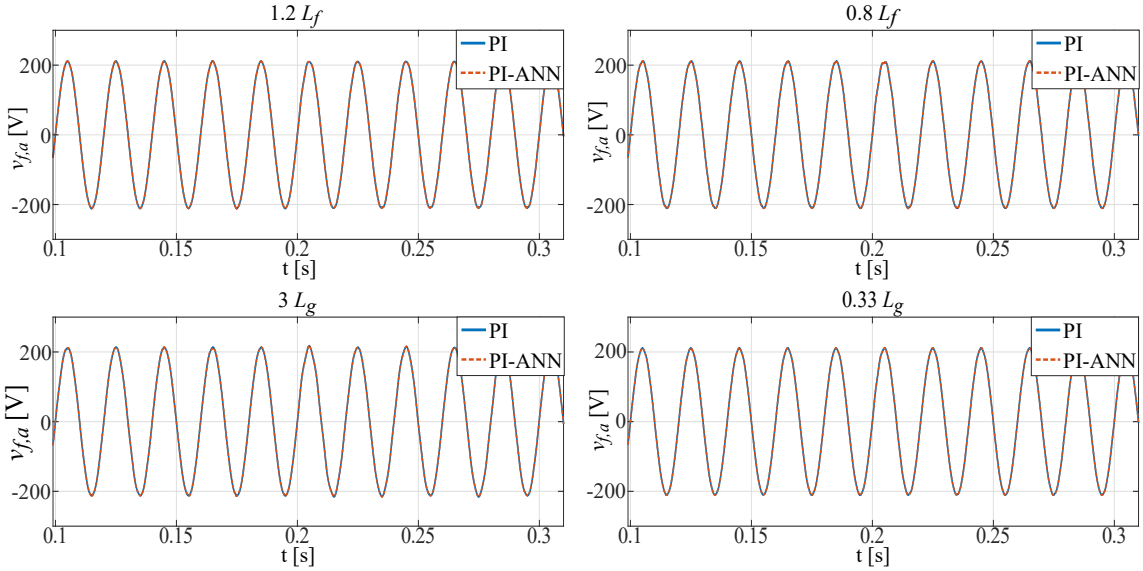


Figure 3.12: Filtered output voltage under the parameter uncertainty condition.

current waveform is observed. Further, there is no significant difference in the $i_{f,a}$ waveform for the different cases. However the current appears slightly more distorted for the low L_f-value,

especially right before and right after the step change. Similar observations are made for the case when the L_g-value is highest. Overall, for all parameter modifications, the performance of the PI controller and the PI-ANN controller is largely comparable.

In Figure 3.12 phase-a of the filtered output voltage is presented. Again, the voltage waveforms appear identical for the two controllers. Further, the voltage waveform appear unaffected by the different modifications. However, looking closely right after the reference current step change at 0.2s, some slight distortions are seen for the decreased filter inductance, the same observation can be made for the increased grid inductance.

3.4 Experimental Results

The performance of the controllers are further verified using the HIL simulation tool OPAL RT. The HIL simulation results are presented for two test cases. The first case considers a reference current step change and the second case considers a short circuit fault at grid-side. The HIL simulation results will be compared to those obtained in the Matlab/Simulink simulation environment and serves as a validation of the controller performance. The d-axis converter current $i_{c,d}$, reference current i_d^*, phase a of filtered output current $i_{f,a}$ and voltage $v_{f,a}$ are plotted for both controllers and test cases. The set-up is given in Figure 3.13.

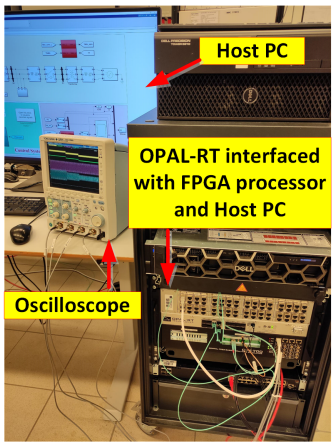
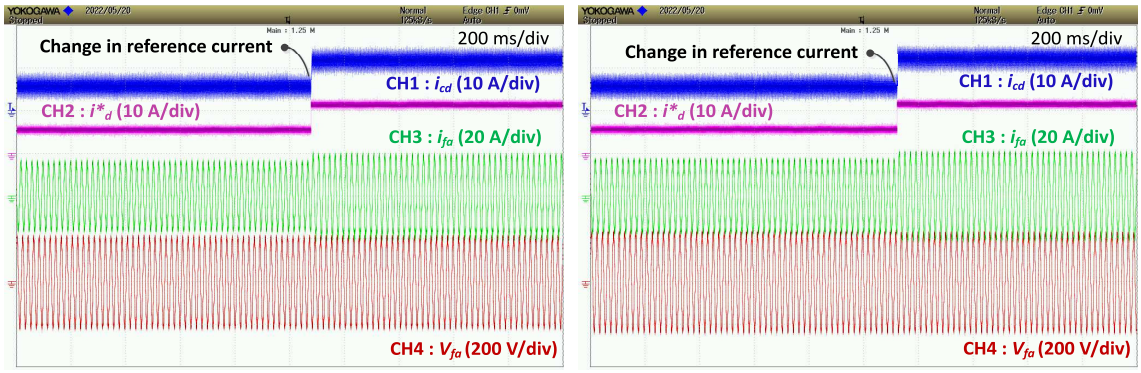


Figure 3.13: Image of the setup of the OPAL RT simulations.

In Figure 3.14 the performance of the PI controller and the PI-ANN controller is presented for the case when the reference current i_d^* is varied from 6A to 12A. In Figure 3.14(a) the PI controller performance is plotted. Here the converter current $i_{c,d}$ is seen to track the step in the i_d^*-value. The filtered output current $i_{f,d}$ and the filtered output voltage $v_{f,d}$ maintain smooth waveforms. This performance corresponds to the observations for the same tracking test case described in Subsection 3.3.1. In Figure 3.14(b) the performance of the PI-ANN controller is presented for the same test case. Also for this controller, the converter current tracks the reference current step, and the current $i_{f,a}$ and the voltage $v_{f,a}$ maintain smooth waveforms. This performance correspond to the observations for the same tracking test case in Subsection 3.3.1.

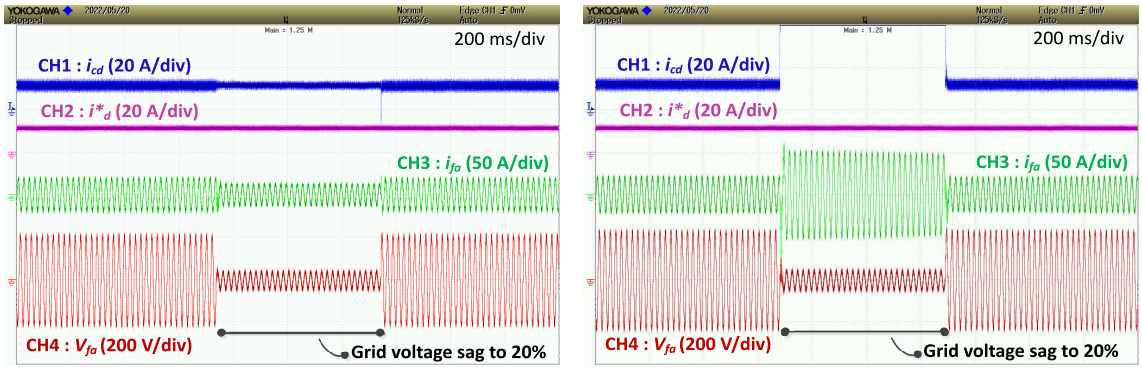
In Figure 3.15 the performance is presented for the test case when the grid voltage drops to 20% while the reference i_d^* is set to 20A. In Figure 3.15(a) the performance of the PI controller is plotted. The converter current remains at the reference value. However, the filtered output current is seen to drop. This was also the case for the simulation results obtained from Matlab/Simulink, however the drop was not as significant. In both simulation environments the output current becomes distorted when the grid voltage drops to 20% and rises back to nominal value. These distortions appear lower and seem to clear faster for the HIL simulation. The filtered output voltage also appear less distorted for the HIL simulation. Overall, a similar performance to that observed in Subsection 3.3.2 is seen for the HIL simulation results. In Figure 3.15(b) the performance of the PI-ANN controller is plotted for the same test case. As was described in Subsection 3.3.2 the

current $i_{c,d}$ does not manage to track the reference when the short circuit is introduced, instead it rises to a significantly higher value. The same behaviour is seen for the current $i_{f,a}$ which has an increased magnitude for the duration of the short circuit. Similar distortions, as observed from the Matlab/Simulink simulations results, are also present at the introduction and clearing of the short circuit. The voltage $v_{f,a}$ is also distorted at the start and end of the short circuit. Overall the HIL-simulated performance corresponds to that observed in the Matlab/Simulink simulation environment.



(a) PI controller (b) PI-ANN controller

Figure 3.14: OPAL RT simulation results for current reference step change.



(a) PI controller (b) PI-ANN controller

Figure 3.15: OPAL RT simulation results for grid-side short circuit fault.

3.5 Discussion

The simulation results from both the Matlab/Simulink environment and the OPAL RT environment show that the PI controller and the PI-ANN controller have a comparable performance in most scenarios. However, in the short circuit fault test case the network performed worse than its expert. This might have been improved by increasing the data set such that the network has more extensive training for similar situations, however, there will always be unobserved situations and this is perhaps not an efficient remedy to the problem. Another remedy is to change the training settings, there are numerous of parameters that must be determined such as hidden layer size, number of epochs, size of validation and testing data sets, learning rate and much more. The goal of the PI-ANN controller should be to exhibit a performance which is comparable to its expert, in all test cases.

4

MPC-Trained Artificial Neural Network Control

In this chapter the MPC-trained artifical neural network (MPC-ANN) controller will be developed. This chapter starts with introducing the MPC in Section 4.1, which includes a description of the operating principle and of the various MPC types. Next, Section 4.2 describes the design of the continuous control set MPC-type (CCS-MPC). In this thesis, the CCS-MPC represents the expert which will later be used for training the network whose design is described in Section 4.3. Once the network is fully trained it is implemented as a controller denoted MPC-ANN, replacing the original CCS-MPC. The performance is evaluated in Section 4.4, where both the CCS-MPC and the MPC-ANN are implemented and simulated for three different test cases. These simulation results are verified in Section 4.5, by comparing them to OPAL RT simulations. In Section 4.6 all simulation results will be discussed and remarks are made.

4.1 Model Predictive Control

In recent years MPC has been proposed as a promising control method due to advantages such as improved dynamic performance, easier handling of MIMO cases and increased flexibility compared to conventional methods [8, 9, 100]. Although being an advanced control method, it offers an intuitive control design based on the predictive model and cost function. The MPC principle is to predict the process output at future time instants and compute an optimal control signal which minimizes a cost function while ensuring that system constraints are met, this is illustrated in Figure 4.1.

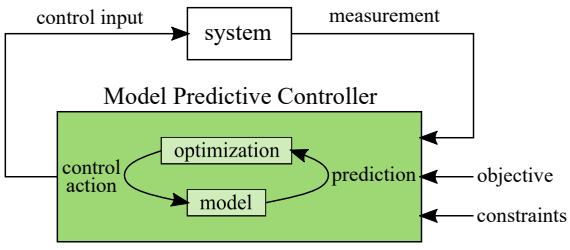


Figure 4.1: Operation principle of model predictive control.

The MPC was first developed in the 80s and started to appear in the industry about 15 years later. However, its entrance into power electronic systems occurred much later, despite being discovered as an advantageous controller already in 1983[101]. The reason was computational speed limitations of the available microprocessors, which was not sufficient for the application of

MPC. In terms of computational power, microprocessors have seen a dramatic improvement and no longer impose limitations, enabling numerous applications of MPC in power electronics [102–104]. Implementing the MPC to improve the operation of grid-connected VSCs is readily applied, and today's research is generally concerned with improving existing MPC methods, making them more robust during grid disturbances and parameter mismatches, in addition to improving the computational efficiency [105–108].

4.1.1 Principle of MPC

The MPC principle is essentially the explicit use of a model to predict the process output at future time instants. The control signals are determined by minimizing an objective function, as given in Equation 4.1, which computes the sum of an objective starting with the current state continuing until the prediction horizon. The minimization of the objective function yields a sequence of optimal control signals, as given by Equation 4.2, of which the first signal is sent to the process. The objective is formulated based on the control objective, which in this case is to track a reference trajectory x^*. Therefore we want to minimize the error between the predicted outputs $\hat{x}$ and the reference. The second term is usually added to limit the amount of control effort Δu, where the λ is a weight indicating the importance of this constraint.

$$\min_{\mathbf{u}} J = \sum_{t=0}^{N-1} (x^*(k+1) - \hat{x}(k+1))^2 + \lambda(\Delta u(k))^2 \tag{4.1}$$

$$\mathbf{u} = \begin{bmatrix} u[k] & u[k+1] \dots u[k+N-1] \end{bmatrix} \tag{4.2}$$

The act of predicting several time steps ahead while implementing only the first control signal is called receding horizon. This strategy was partly developed to address two major drawbacks of the linear quadratic regulator (LQR), whose main difference to the MPC is the use of a fixed horizon [109]. When using a fixed horizon the optimization problem is computed at time k=0 and used throughout the entire prediction horizon. If disturbances which were not predicted by or included in the model occur in the process, the fixed control signals become obsolete. Another drawback is that the control usually weakens when approaching the final time step $k = i + N - 1$. This happens independently of the current state of the system, which may deviate significantly from the previous state, because the impact on the cost function is minor with respect to the total impact of all control signals during the prediction horizon. The receding strategy is illustrated in Figure 4.2, at each time step k the MPC solves the optimization problem, obtains the sequence of predicted control signals $u[k]$ until $u[k + N - 1]$ and applies only the first control signal to the process. In contrast to LQR, the use of a receding horizon results in a control which offers both open loop optimization and feedback control [110].

4.1.2 Types of MPC

The MPC strategies for power converters and drives can be broadly classified considering the optimization technique which is either continuous or finite [8, 100, 111]. The resulting classes are called CCS-MPC and FCS-MPC. However, extensive research in the field of MPC has resulted in additional sub-categories. The classification of some of the most popular MPC strategies is given in Figure 4.3.

There are advantages and disadvantages of all methods. There is still ongoing research seeking to improve existing methods, and in the end the choice of the MPC approach will depend on the specifics and objectives of the application [44, 53, 112–114]. In this thesis the CCS-MPC method is implemented and will therefore be further explained in Section 4.2.

The CCS-MPC computes a continuous control signal, these are then sent to a modulator. The modulator generates the switching signals which in turn provide the desired converter voltage. The sub-categories explicit model predictive control (EMPC) and generalized predictive control (GPC) are two of the most applied CCS-MPC methods. In GPC the solution to the minimization problem

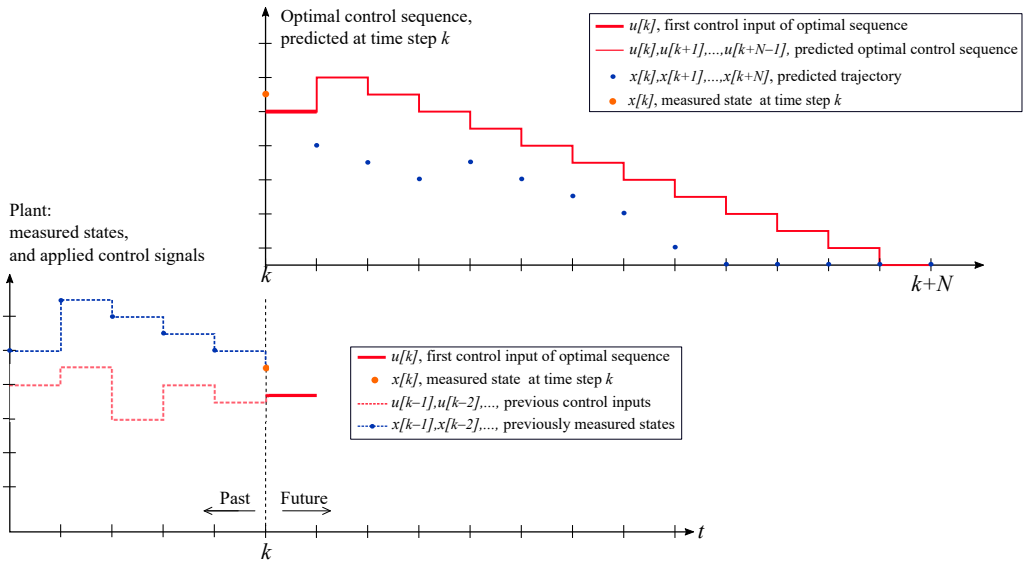


Figure 4.2: Model predictive control strategy, adapted from [110]

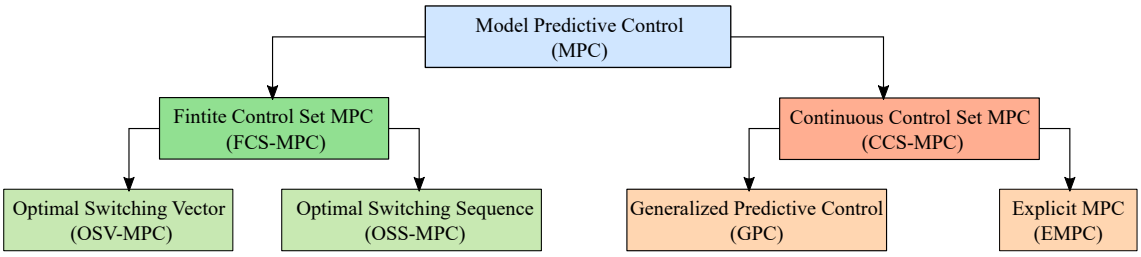


Figure 4.3: Classification of the MPC methods.

is partly computed offline, and is therefore an effective method which limits the computational burden. However the limitation of GPC is that it can only work with linear and unconstrained problems. The EMPC, on the other hand, accepts nonlinear and constrained systems. In this method the whole solution to the minimization problem is stored offline. As a result the online optimization consists of searching through already-stored solutions.[8]

The FCS-MPC methods take advantage of the fact that a converter has a fixed set of switching states. Therefore, at each sampling instant the FCS-MPC predicts the system response for each switching state, evaluates the cost function for each response, and finally delivers the control sequence which minimizes this cost. Since the optimal switching signals are computed in the minimization process these are sent directly to the converter, without the use of an external modulator. FCS-MPC can be further classified as either optimal switching vector (OSV) MPC or optimal switching sequence (OSS) MPC. OSV-MPC uses the output voltage vectors of the converter, previously defined by Equation 2.3, as the control set. This results in a control set which consists of total $2^3 = 8$ voltage vectors previously presented in Figure 2.6. The predictions are calculated for only the voltage vectors in the control set, as a result the minimization problem is reduced to a search algorithm, where the switching states that produces the voltage vector which minimize the objective function are chosen. The main limitation of this strategy is that only one voltage vector is applied during each switching period, and unless constraints are added one voltage vector may be repeated in consecutive periods which generates a variable switching frequency. The OSS-MPC method, on the other hand, considers a control set which consists of possible switching sequences. Therefore in each switching period, several switching states, i.e. voltage vectors, are applied.

4.2 Continuous Control Set MPC

In this section the CCS-MPC method will be developed and described. The section goes step-wise through all necessary mathematical equations and provides an in-depth description of the control development. The following design approach is based on the articles [12] and [115].

4.2.1 Analytical Model

The CCS-MPC when implemented as the inner current controller for the grid-connected VSC is illustrated in Figure 4.4. The optimal control signal $\boldsymbol{v}^*_{c,\alpha\beta}$ is calculated by minimizing an objective function which is based on the system model and future predictions. Once the optimal control sequence is determined only the first signal is sent to the pulse-width modulator which generates the switching commands.

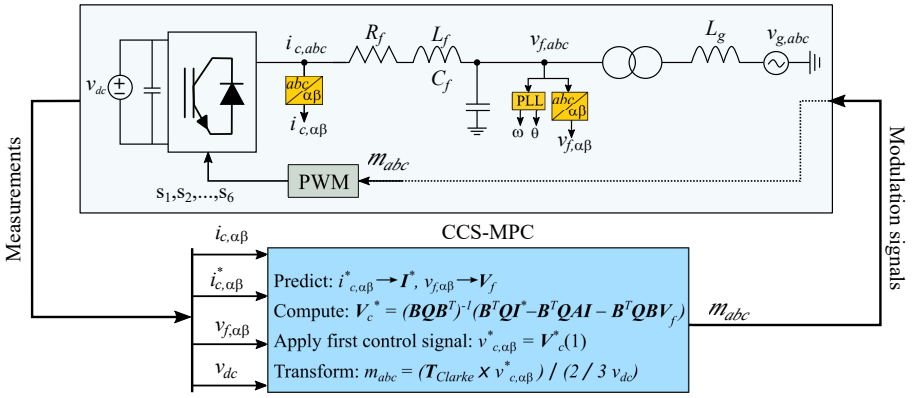


Figure 4.4: Implementation of CCS-MPC as an inner current controller for the grid-connected VSC.

4.2.2 Discretization

The design of a single horizon non-constrained CCS-MPC is intuitive and serves as a basis for extending the horizon. As for the PI controller, the control method is model-based and thus depends on the system dynamics. The controller will be implemented in the stationary ($\alpha\beta$) reference frame and the system, where the system dynamics are described by Equation 4.3.

$$\boldsymbol{v}_{c,\alpha\beta} = R_f \boldsymbol{i}_{c,\alpha\beta} + L_f \frac{d\boldsymbol{i}_{c,\alpha\beta}}{dt} + \boldsymbol{v}_{f,\alpha\beta} \tag{4.3}$$

Rearranging in terms of the derivative of the current with respect to time, and using state-space formulation results in Equation 4.4.

$$\frac{d}{dt}\underbrace{\begin{bmatrix} i_{c,\alpha} \\ i_{c,\beta} \end{bmatrix}}_{\boldsymbol{i}_{c,\alpha,\beta}} = \underbrace{\begin{bmatrix} -\frac{R_f}{L_f} & 0 \\ 0 & -\frac{R_f}{L_f} \end{bmatrix}}_{\mathbf{A}_c} \begin{bmatrix} i_{c,\alpha} \\ i_{c,\beta} \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{1}{L_f} & 0 \\ 0 & \frac{1}{L_f} \end{bmatrix}}_{\mathbf{B}_c} \underbrace{\begin{bmatrix} v_{c,\alpha} - v_{f,\alpha} \\ v_{c,\beta} - v_{f,\beta} \end{bmatrix}}_{\boldsymbol{u}_{\boldsymbol{\alpha},\boldsymbol{\beta}}} \tag{4.4}$$

This model is continuous, but as the implementation of the MPC is in a computer the model must be in discrete form, therefore the continuous system must be discretized. There are several discretization methods, here it is chosen to use the *exact discretization* method. This is based on the exact solution of the LTI system which is given by Equation 4.5.

$$\boldsymbol{x}(t) = e^{\mathbf{A}t}\boldsymbol{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\boldsymbol{u}d\tau \tag{4.5}$$

The exact solution is discretized using the substitutions $x[k] = x(kT)$ and $x[k+1] = x((k+1)T)$ and assuming that the control input $\boldsymbol{u}$ remains constant between samples, $\int_{kT}^{(k+1)T} \boldsymbol{u}(\tau)d\tau = \boldsymbol{u}[k]$. This results in Equations 4.6a, 4.6b and 4.6c. Where the substitution $v(\tau) = kT + T - \tau$ and $d\tau = -dv$ is used in Equation 4.6c .

$$\boldsymbol{x}[k] = e^{\mathbf{A}kT}\boldsymbol{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}\boldsymbol{u}d\tau \tag{4.6a}$$

$$\begin{aligned}\boldsymbol{x}[k+1] &= e^{\mathbf{A}(k+1)T}\boldsymbol{x}(0) + \int_0^{(k+1)T} e^{\mathbf{A}((k+1)T-\tau)}\mathbf{B}\boldsymbol{u}d\tau \\ &= e^{\mathbf{A}T}\underbrace{\left(e^{\mathbf{A}kT}\boldsymbol{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}\mathbf{B}\boldsymbol{u}d\tau\right)}_{\boldsymbol{x}[k]} + \int_{kT}^{(k+1)T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}\boldsymbol{u}d\tau\end{aligned} \tag{4.6b}$$

$$\begin{aligned}\boldsymbol{x}[k+1] &= e^{\mathbf{A}T}\boldsymbol{x}[k] - \left(\int_{v(kT)}^{v(k+1)T} e^{\mathbf{A}v}dv\right)\mathbf{B}\boldsymbol{u}[k] \\ &= e^{\mathbf{A}T}\boldsymbol{x}[k] + \left(\int_0^{T} e^{\mathbf{A}v}dv\right)\mathbf{B}\boldsymbol{u}[k] \\ &= e^{\mathbf{A}T}\boldsymbol{x}[k] + \mathbf{A}^{-1}(e^{\mathbf{A}T} - \mathbf{I})\mathbf{B}\boldsymbol{u}[k]\end{aligned} \tag{4.6c}$$

The discretized system is therefore given by Equation 4.7, where the discrete matrices $\mathbf{A}$ and $\mathbf{B}$ now replace the continuous $\mathbf{A}_c$ and $\mathbf{B}_c$.

$$\boldsymbol{i}_{c,\alpha\beta}[k+1] = \underbrace{e^{\mathbf{A}_c T}}_{\mathbf{A}}\boldsymbol{i}_{c,\alpha\beta}[k] + \underbrace{\mathbf{A}_c^{-1}\left(e^{\mathbf{A}_c T} - \mathbf{I}\right)\mathbf{B}_c}_{\mathbf{B}}(\boldsymbol{v}_{c,\alpha\beta} - \boldsymbol{v}_{f,\alpha\beta}) \tag{4.7}$$

4.2.3 Cost Function

The CCS-MPC is implemented as the inner current controller and should at all times track the reference current. Therefore a suitable objective is to minimize the deviation between the measured converter current and its reference, this gives the objective function in Equation 4.8. The minimization yields an optimal converter voltage $\boldsymbol{v}_{c,\alpha\beta}$ which is later transformed to *abc*-components and sent to the modulator. The weight matrix $\mathbf{Q}$ is included to allow for scaling of the importance of the α and β components.

$$\min_{\boldsymbol{v}_{c,\alpha\beta}} C = (\boldsymbol{i}^*_{\alpha\beta} - \boldsymbol{i}_{c,\alpha\beta}[k+1])^\top \mathbf{Q}(\boldsymbol{i}^*_{\alpha\beta} - \boldsymbol{i}_{c,\alpha\beta}[k+1]) \tag{4.8}$$

The current $\boldsymbol{i}_{c,\alpha\beta}[k+1]$ is a function of the voltage $\boldsymbol{v}_{c,\alpha\beta}$ as given by Equation 4.7, therefore minimizing C consists of determining the voltage $\boldsymbol{v}_{c,\alpha\beta}$ at which the current deviation becomes minimal. In other words, by solving $\frac{\partial C}{\partial \boldsymbol{v}_{c,\alpha\beta}} = 0$ and inserting the model (4.7) the optimal voltage is is expressed by Equation 4.9.

$$\boldsymbol{v}^*_{c,\alpha\beta}[k] = \left(\mathbf{B}\mathbf{Q}\mathbf{B}^\top\right)^{-1}\left(\mathbf{B}^\top\mathbf{Q}\boldsymbol{i}^*_{\alpha\beta} - \mathbf{B}^\top\mathbf{Q}\mathbf{A}\boldsymbol{i}_{c,\alpha\beta}[k+1] - \mathbf{B}^\top\mathbf{Q}\mathbf{B}\boldsymbol{v}_{f,\alpha\beta}[k]\right) \tag{4.9}$$

In Equation 4.10 the expression is further simplified, here the weight $\mathbf{Q}$ in Equation 4.11 is chosen, and the simplification is based on the fact that all matrices are symmetric,i.e. $\mathbf{M}^\top = \mathbf{M}$ and that $\mathbf{B}^\top\mathbf{B}$ is invertible.

$$\boldsymbol{v}_{c,\alpha\beta}[k] = \mathbf{B}^{-1}\boldsymbol{i}^*_{\alpha\beta} - \mathbf{B}^{-1}\mathbf{A}\boldsymbol{i}_{\alpha\beta}[k+1] - \boldsymbol{v}_{\alpha\beta}[k] \tag{4.10}$$

$$\mathbf{Q} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \tag{4.11}$$

4.2.4 Extended Horizon

If a single horizon is used, then the optimal voltage in Equation 4.10 may be directly implemented in the controller. For a prediction horizon of N steps the system must be augmented. This augmentation results in new states and inputs which now consists of the previous states and inputs predicted until time step N. This is given by Equations 4.12.

$$\begin{aligned} \boldsymbol{I} &= \begin{bmatrix} \boldsymbol{i}_{c,\alpha\beta}[k+1] & \boldsymbol{i}_{c,\alpha,\beta}[k+2] & \dots & \boldsymbol{i}_{c,\alpha\beta}[k+N] \end{bmatrix}^\top \\ \boldsymbol{U} &= \boldsymbol{V}_c - \boldsymbol{V}_f = \begin{bmatrix} \boldsymbol{u}_{\alpha\beta}[k] & \boldsymbol{u}_{\alpha\beta}[k+1] & \dots & \boldsymbol{u}_{\alpha\beta}[k+N-1] \end{bmatrix}^\top \end{aligned} \tag{4.12}$$

The system matrices are correspondingly augmented by representing the state at every future time step $k+1$, $k+2$, ...$k+N$ as a function of the states of all previous time steps, starting with the current sample k, this is easier understood by Equation 4.13.

$$\begin{aligned} \boldsymbol{i}_{c,\alpha\beta}[k+1] &= \mathbf{A}\boldsymbol{i}_{c,\alpha\beta}[k] + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k] \\ \boldsymbol{i}_{c,\alpha\beta}[k+2] &= \mathbf{A}\boldsymbol{i}_{c,\alpha\beta}[k+1] + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k+1] \\ &= \mathbf{A}\left(\mathbf{A}\boldsymbol{i}_{c,\alpha\beta}[k] + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k]\right) + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k+1] \\ &= \mathbf{A}^2\boldsymbol{i}_{c,\alpha\beta}[k] + \mathbf{AB}\boldsymbol{u}_{\alpha\beta}[k] + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k+1] \\ \boldsymbol{i}_{c,\alpha\beta}[k+3] &= \mathbf{A}\boldsymbol{i}_{c,\alpha\beta}[k+2] + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k+2] \\ &= \mathbf{A}\left(\mathbf{A}^2\boldsymbol{i}_{c,\alpha\beta}[k] + \mathbf{AB}\boldsymbol{u}_{\alpha\beta}[k] + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k+1]\right) + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k+2] \\ &= \mathbf{A}^3\boldsymbol{i}_{c,\alpha\beta}[k] + \mathbf{A}^2\mathbf{B}\boldsymbol{u}_{\alpha\beta}[k] + \mathbf{AB}\boldsymbol{u}_{\alpha\beta}[k+1] + \mathbf{B}\boldsymbol{u}_{\alpha\beta}[k+2] \end{aligned} \tag{4.13}$$

Using the augmented states, inputs and the augmentation technique in Equation 4.13, the system matrices in Equation 4.14 are obtained.

$$\begin{aligned} \mathbf{A}_{aug} &= \begin{bmatrix} \mathbf{A} & \mathbf{A}^2 & \dots & \mathbf{A}^N \end{bmatrix}^\top \\ \mathbf{B}_{aug} &= \begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{AB} & \mathbf{B} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{A}^2\mathbf{B} & \mathbf{AB} & \mathbf{B} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{A}^{N-1}\mathbf{B} & \mathbf{A}^{N-2}\mathbf{B} & \dots & & \mathbf{AB} & \mathbf{B} \end{bmatrix} \end{aligned} \tag{4.14}$$

The objective function previously given in Equation 4.1 remains the same and solving for the optimal voltage now consists of solving for the augmented voltage vector

$\boldsymbol{V}_c = \begin{bmatrix} \boldsymbol{v}_{c,\alpha,\beta}[k] & \boldsymbol{v}_{c,\alpha,\beta}[k+1] & \dots & \boldsymbol{v}_{c,\alpha,\beta}[k+N-1] \end{bmatrix}$.

The corresponding expression is given by Equation 4.15

$$\boldsymbol{V}_c^* = (\mathbf{B}_{aug}\mathbf{Q}_{aug}\mathbf{B}_{aug}^\top)^{-1}(\mathbf{B}_{aug}^\top\mathbf{Q}\boldsymbol{I}^* - \mathbf{B}_{aug}^\top\mathbf{Q}_{aug}\mathbf{A}_{aug}\boldsymbol{I} - \mathbf{B}_{aug}^\top\mathbf{Q}\mathbf{B}_{aug}\boldsymbol{V}_f) \tag{4.15}$$

4.2.5 Prediction Model

The final step is to define the predicted currents $\boldsymbol{i}_{c,\alpha\beta}[k+t]$ and filtered output voltages $\boldsymbol{v}_{f,\alpha\beta}[k+t-1]$ needed for computing the optimal predicted voltage vector $\boldsymbol{V}_c^*$ in Equation 4.15. The prediction is identical for the voltage and current, and will thus only be shown for the voltage. Starting with the definition of the derivative

$$\frac{d}{dt}\boldsymbol{v}_{f,\alpha\beta}[k] = \frac{\boldsymbol{v}_{f,\alpha\beta}[k+1] - \boldsymbol{v}_{f,\alpha\beta}[k]}{T_s} \tag{4.16}$$

where the voltage in $\alpha\beta$-frame is defined by an amplitude U_f and a phase angle θ given by

$$\begin{aligned} v_{f,\alpha}[k] &= U_f \cos\theta \\ v_{f,\beta}[k] &= U_f \sin\theta \end{aligned} \tag{4.17}$$

Deriving the voltages, and noting that $\frac{d\theta}{dt} = \omega t$, gives the following relations

$$\begin{aligned} \frac{d}{dt} v_{f,\alpha}[k] &= -\omega U_f \sin \omega t = -\omega U_f \sin \theta = -\omega v_{f,\beta}[k] \\ \frac{d}{dt} v_{f,\beta}[k] &= \omega U_f \cos \omega t = \omega U_f \cos \theta = \omega v_{f,\alpha}[k] \end{aligned} \tag{4.18}$$

rearranging (4.16) and inserting (4.18) results in the following predicted behaviour

$$\begin{aligned} v_{f,\alpha}[k+1] &= v_{f,\alpha}[k] - \omega T_s v_{f,\beta}[k] \\ v_{f,\beta}[k+1] &= v_{f,\beta}[k] + \omega T_s i_\alpha[k] \end{aligned} \tag{4.19}$$

The CCS-MPC algorithm and the development of matrices for a single and an extended horizon is given in Appendix E.

4.2.6 Determining the Prediction Horizon

The determination of the prediction horizon was done by comparing the controller performance for different horizons. Increasing it to 4 time steps gave the best performance, while increasing it further resulted in an equivalent performance. Therefore a prediction horizon of 4 time steps was chosen. A comparison of the produced d-axis converter current $i_{c,d}$ during a reference step from 5A to 20A is plotted for the CCS-MPC with a single horizon ($N_p = 1$) and with a 4-step prediction horizon (N_p=4) in Figure 4.5. Although the performance of the two controllers is largely comparable, it is seen that the deviation between $i_{c,d}$ and i_d^* is reduced when the controller with a 4-steps prediction horizon is employed.

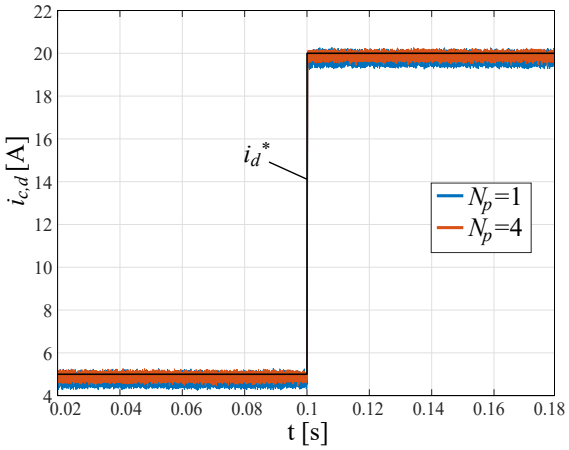


Figure 4.5: Comparison of controller tracking performance during a reference step change from 5A to 20A, using a single step horizon and a 4-step prediction horizon.

4.3 Network Design

The MPC-ANN consists of a feedforward neural network (FFNN) which is trained using the CCS-MPC as an expert. Similar to the PI-ANN this network will first be trained according to the LM-algorithm described in Subsection 3.2.1, using the simulation data obtained from the CCS-MPC, before replacing the original controller, this is illustrated in Figure 4.6.

The development of the network controller consists of choosing an appropriate network structure. The number of inputs and outputs are in this case chosen equal to the inputs and outputs of the MPC-block shown in Figure 4.4 and Figure 4.6 for the offline training, this results in 7 inputs ($i_{\alpha\beta}$, $i^*_{\alpha\beta}$, $v_{\alpha\beta}$ and v_{dc}) and 3 outputs (m_{abc}). The hidden layer size is determined based on the

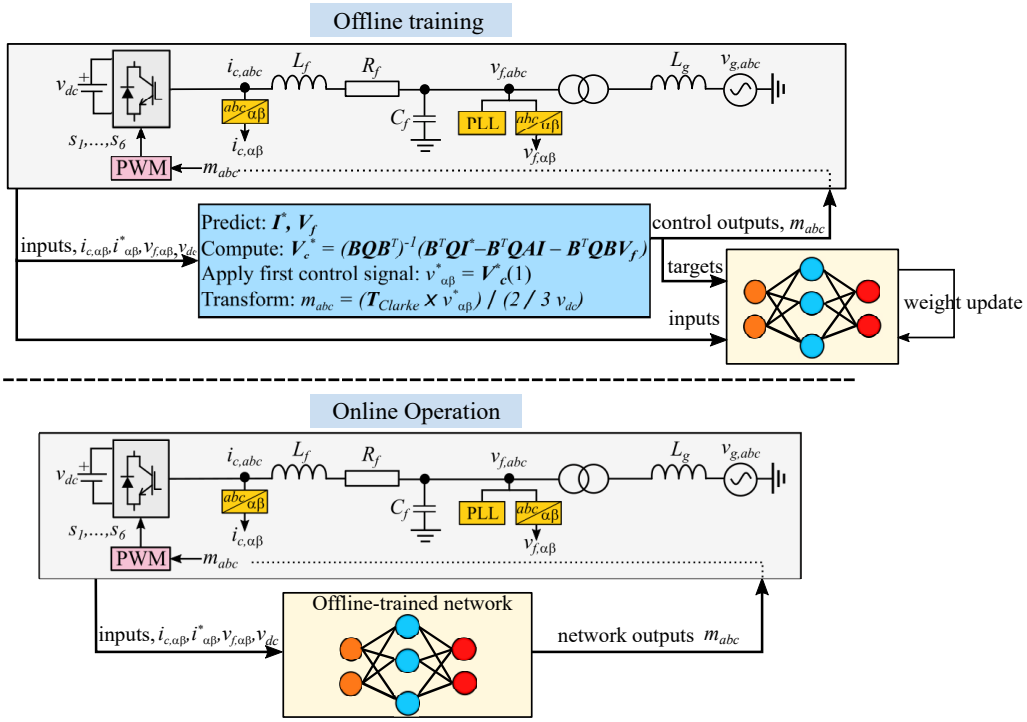


Figure 4.6: MPC-ANN control design.

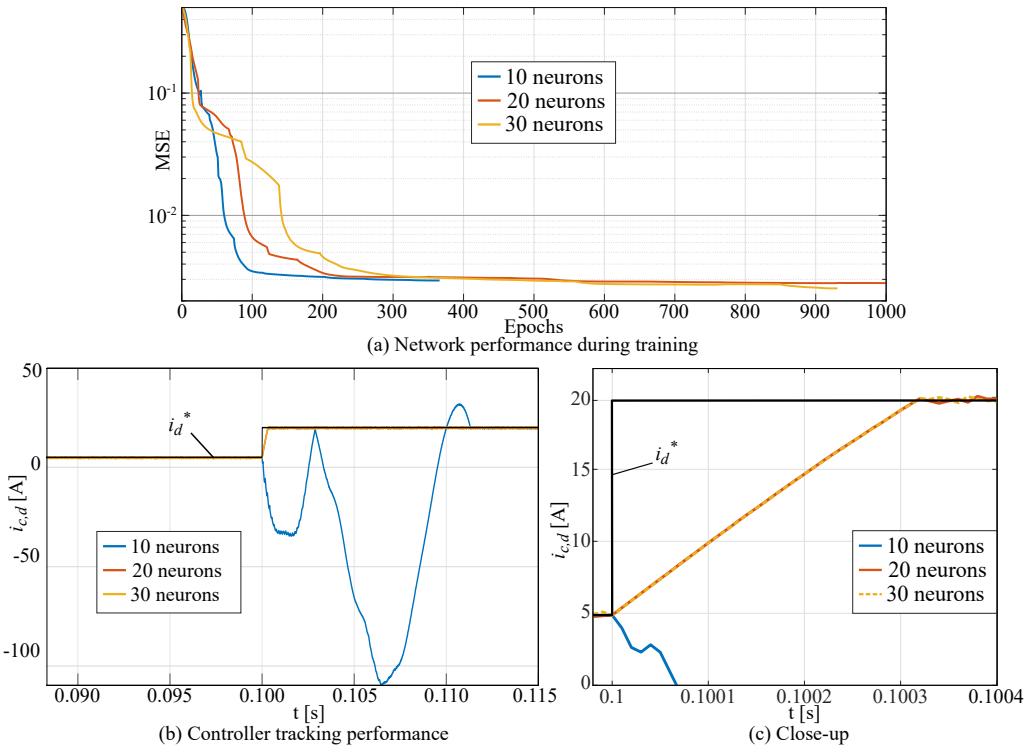
performance of the trained network once implemented. Since having only one layer gave satisfactory results it was not considered to increase the number of hidden layers.

The network performance during training for three different hidden layer sizes is presented in Figure 4.7(a). Here the hidden layers consist of 10, 20 and 30 neurons. For the network with 10 hidden neurons training was stopped at 366 epochs, due to validation errors. See Section 2.3.3 for further information on the validation-procedure. The same situation is seen for the network with 30 hidden neurons at 925 epochs. The network with 10 hidden neurons finishes training and the best performance is seen to be at the final epoch.

The tracking performance of the network when implemented as a controller during a step from 5A to 20A at 0.2s is presented in Figure 4.7(b) and (c) for the same three hidden layer sizes as in Figure 4.7(a). When the reference step occurs, the network controller with 10 hidden neurons produces an $i_{c,d}$-value which deviates significantly from the reference i_d^* for a time period of 10 ms. The network controllers trained using 20 and 30 hidden neurons have a better performance. From the close-up in Figure 4.7(c) the performance of the these two network controllers seems almost identical. Since it is always recommended to keep the network structure as small as possible, the structure 7-20-3 is chosen for the finalized MPC-ANN controller.

4.4 Performance Evaluation

Both the CCS-MPC and the MPC-trained ANN are implemented in the Matlab/Simulink environment, for a switching model of the VSC. The controller parameters are given in Table 4.1, and the system parameters are the same as given in Table 3.1. The implementation of the CCS-MPC is illustrated in Figure 4.4, and the implementation of the MPC-ANN controller is illustrated in Figure 4.6. The performance of both controllers is evaluated for three different control scenarios which will be described in the following subsections.



(a) Network performance during training

(b) Controller tracking performance

(c) Close-up

Figure 4.7: Network performance during training and controller tracking performance during simulations for different network structures.

Table 4.1: CCS-MPC and MPC-ANN controller parameters.

Symbol	Description	Value
N_p	Prediction horizon	4
	Network structure	7-20-3
$Epoch^{max}$	Maximum number of epochs	1000
μ_0	Blending factor	0.01
μ^{max}	Maximum blending factor	10^{10}
μ_{inc},μ_{dec}	Increase/decrease of blending factor	10
$\frac{\partial C}{\partial x}^{min}$	Minimum gradient	10^{-7}
E_{val}	Maximum validation errors	6

4.4.1 Current Reference Tracking

In this test case a simulation is carried out for both the CCS-MPC and the MPC-ANN controllers when varying the d-axis reference current magnitude (i_d^*) continuously and step-wise. The obtained results under this test case are presented in Figure 4.8. Here the successful implementation of both controllers can be verified by observing the measured d-axis current ($i_{c,d}$) which is accurately tracking the corresponding reference current. There is no visible difference in the tracking performance of the two controllers. Phase-a of the output current $i_{f,a}$ and output voltage $v_{f,a}$ are also included. It can be observed that both controllers maintain a stable and sinusoidal $i_{f,a}$ with small distortions. However, the current has a consistently lower magnitude and is slightly shifted when the MPC-ANN is implemented compared to the MPC. The $v_{f,a}$ waveform is satisfactorily maintained at the desired level by both controllers.

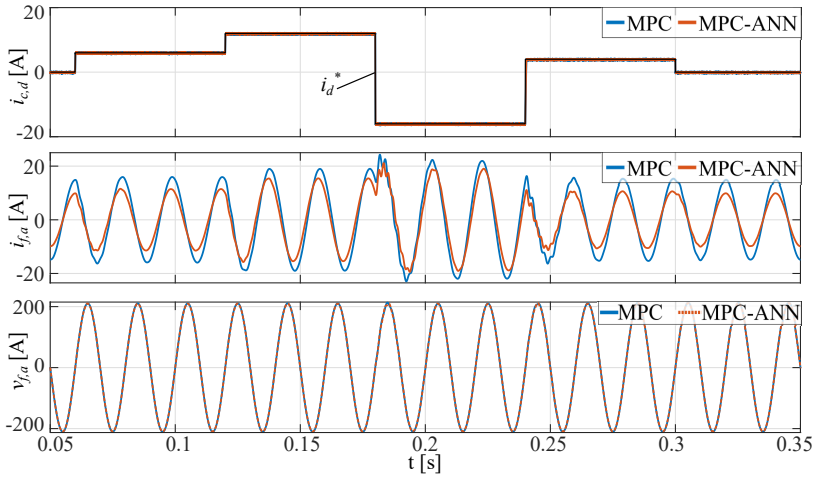


Figure 4.8: Performance evaluation of the CCS-MPC and the MPC-ANN controllers for a varying reference current.

4.4.2 Analysis During Grid-Side Fault

The performance of the CCS-MPC and the MPC-ANN is tested under grid-side fault conditions by reducing the grid voltage amplitude $|v_g|$ symmetrically to 20% of its nominal value. The fault is introduced at 0.2s and lasts for 3 cycles, while the reference current magnitude i_d^* is kept constant at 18A. The resulting d-axis converter current $i_{c,d}$, phase-a of filtered output current $i_{f,a}$ and filtered output voltage $v_{f,a}$ are presented in Figure 4.9. It is seen that when the short circuit is introduced there is a slight increase in deviation between $i_{c,d}$ and i_d^*. This lasts until the short circuit clears at 0.26s. At the moment of clearing both controllers produce a converter current which has a significant drop at the moment of clearing. Where the $i_{c,d}$-value drops from 18A to ∼6A and ∼1A, for the MPC and the MPC-ANN, respectively. Both controllers produce an output current $i_{f,a}$ with a smooth waveform. However, some distortion appears at the instants when the short circuit is introduced and when it is cleared. The distortion seems somewhat greater for the MPC. The waveform of the output voltage $v_{f,a}$ is smooth for both controllers.

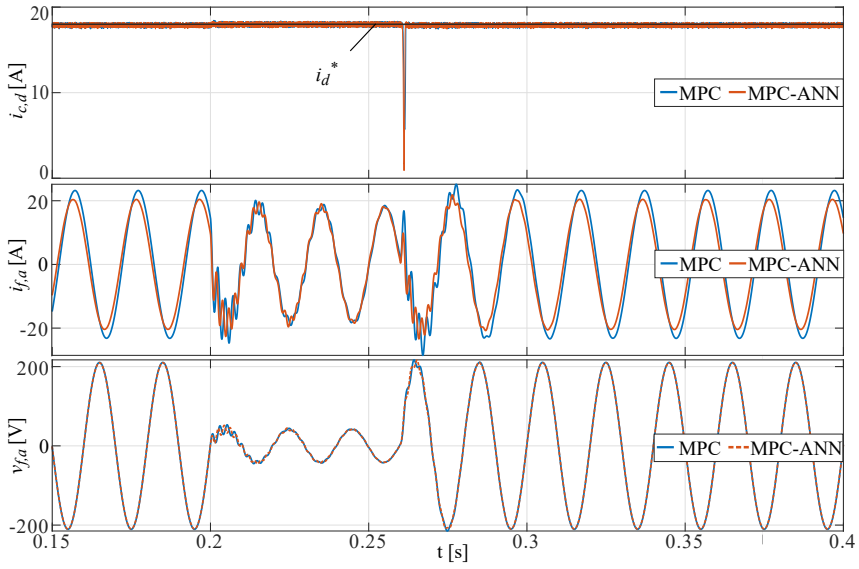


Figure 4.9: Performance evaluation of the CCS-MPC and the MPC-ANN controllers under grid-fault.

4.4.3 Parameter Uncertainty

The robustness of the CCS-MPC and the MPC-ANN is further verified under the parameter uncertainty test case. Now the controller performance is evaluated for a 20% increased and decreased filter inductance L_f and a 300% increased and decreased grid inductance L_g. In addition, the reference current i_d^* is changed from 4A to 18A at 0.2s. In Figure 4.10, the converter current $i_{c,d}$ and the reference current i_d^* are presented for both controllers. It is seen that the performance of the CCS-MPC and the MPC-ANN are largely comparable and appear unaffected by the modifications. Except when the L_f-value is decreased, now both controllers produce a converter current which deviates slightly more from the reference, than what is observed for the other parameter modifications. It was also observed a slightly faster response for the MPC-ANN for the low L_f-value, and a slightly slower response for the high L_g-value. These observations are visible in the close-up in the bottom right corner. However, note the short timescale of 0.2s-0.2005s and the very small response speed difference. In Figure 4.11 phase a of the filtered output current is presented for the same test case as above using both controllers. When the MPC-ANN controller is employed the $i_{f,a}$-value settles at a lower value. Despite this, the performance of the controllers are very similar and both produce a smooth output current which appears unaffected by the parameter modifications. In Figure 4.12 phase a of the filtered output voltage is presented for the parameter uncertainty test case. The voltage waveform is smooth for both controllers and appears unaffected for all parameter modifications. There is no visible difference in the performance of the two controllers.

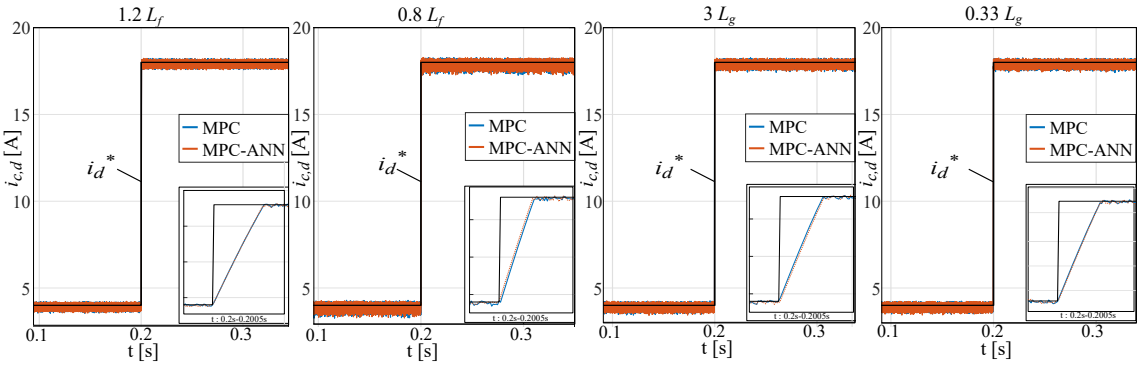


Figure 4.10: D-axis converter current for parameter uncertainty test case.

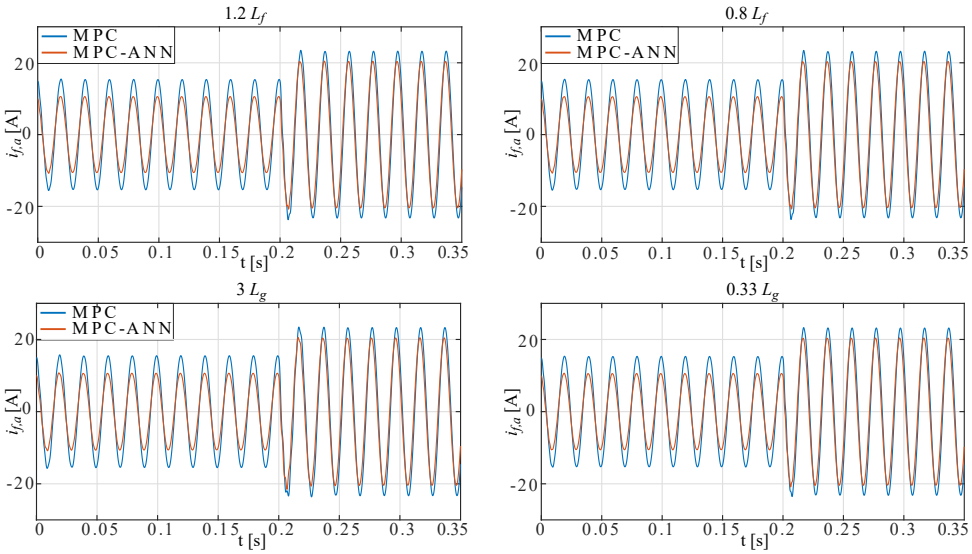


Figure 4.11: Phase a of filtered output current for parameter uncertainty test case.

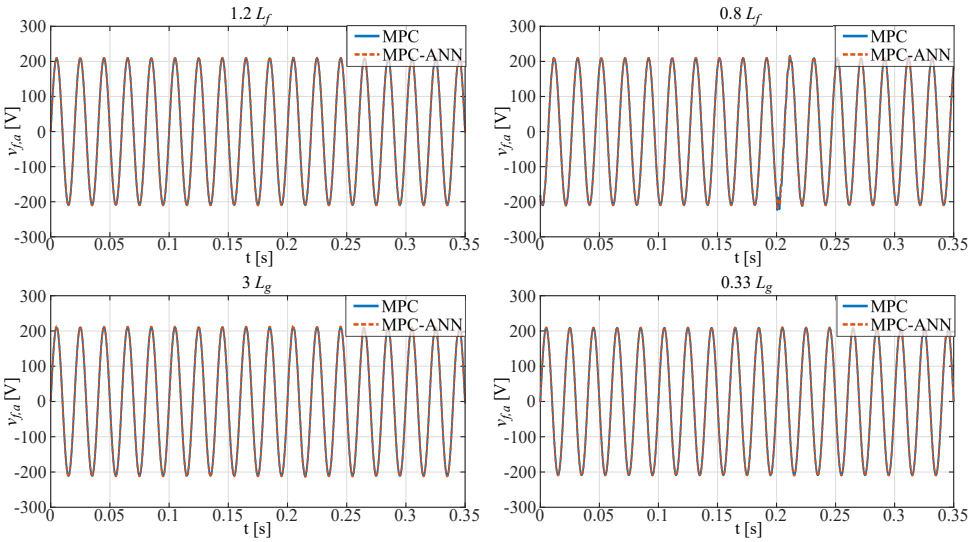
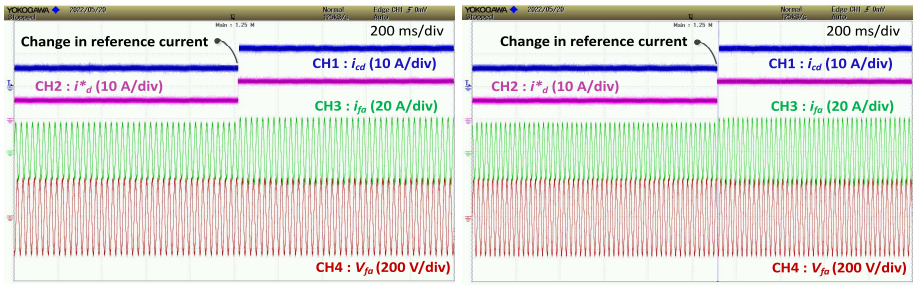


Figure 4.12: Phase a of filtered output voltage for parameter uncertainty test case.

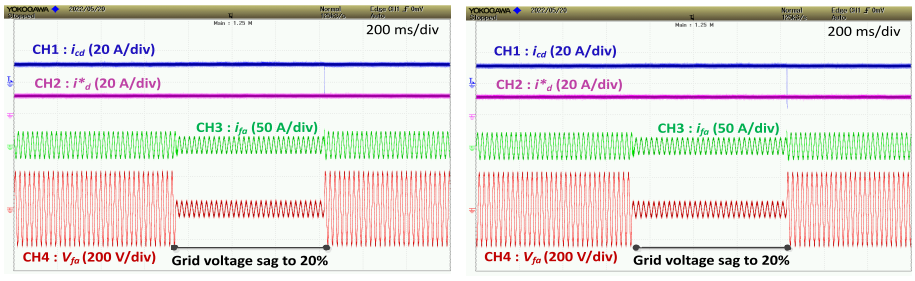
4.5 Experimental Results

The performance of the CCS-MPC and the MPC-ANN controllers is verified using the HIL simulation tool OPAL RT. The HIL simulation results are presented for two test cases: first, for a step change in reference current. Secondly for a short circuit fault introduced at the distribution grid. The following HIL simulation results will be compared to those obtained in the Matlab/Simulink simulation environment and serves as a validation of the controller performance. The d-axis converter current $i_{c,d}$, reference current i_d^*, phase-a of filtered output current $i_{f,a}$ and voltage $v_{f,a}$ are plotted for both controllers and test cases.



(a) MPC (b) MPC-ANN

Figure 4.13: OPAL RT simulation results for current reference step change.



(a) MPC (b) MPC-ANN

Figure 4.14: OPAL RT simulation results for grid-side short circuit fault.

In Figure 4.13 the performance of both controllers for a step change in the reference current i_d^* is presented. The i_d^*-value is changed from 6A to 12A. In Figure 4.13(a) the performance of the

CCS-MPC is presented. The converter current $i_{c,d}$ successfully manages to track the reference change, and the waveform of both the current $i_{f,a}$ and the voltage $v_{f,a}$ appear smooth and has an expected amplitude value. In Figure 4.13(b) the performance of the MPC-ANN controller is presented for the same test case. The same observations as for the CCS-MPC is made here, the current $i_{c,d}$ manages to track the reference current i_d^*, and the current $i_{f,a}$ and voltage $v_{f,a}$ has smooth waveforms with expected amplitude values. In conclusion, the OPAL RT simulated performance of the CCS-MPC and MPC-ANN for a change in current reference appear similar to the results obtained from the Matlab/Simulink simulations.

In Figure 4.14 the performance of both controllers is presented for the short circuit test case. The short circuit is represented by a grid voltage drop to 20%. In Figure 4.14(a) the performance of the CCS-MPC is plotted. Similar d-axis current as in Subsection 4.4.2 is observed, where no impact is seen when the short circuit is introduced and a large drop occurs when the short circuit is cleared. The current $i_{f,a}$ is distorted at the introduction and clearing of the short circuit, but has a smooth waveform otherwise. The voltage $v_{f,a}$ also has a smooth waveform, but appears more distorted during the short circuit. In Figure 4.14(b) the performance of the MPC-ANN controller is presented. Here similar observations are made. The $i_{c,d}$-value has a significant drop at short circuit clearing, but appears unaffected when the short circuit is introduced. The waveform of the current $i_{f,a}$ and the voltage $v_{f,a}$ are smooth but appear slightly distorted during the short circuit. In conclusion, the overall performance of both controllers are comparable to the observations made from the Matlab/Simulink simulation environment.

4.6 Discussion

The simulation results from both the Matlab/Simulink environment and the OPAL RT environment show that the CCS-MPC controller and the MPC-ANN controller have a comparable performance for all test case scenarios. Both exhibit accurate tracking, fast dynamic response and robustness against parameter modifications. However, in terms of computational burden, the ANN-based controller becomes the best choice [53]. This was also one of the stated advantages in [6], where an FCS-MPC was used as the expert for training the neural network. The FCS-method has less computational complexity than the CCS-MPC method, and therefore the benefit of the MPC-ANN controller is most likely larger in this project.

5

Recurrent Neural Network-Based Control

In this chapter an RNN-based controller will be developed, this control method is based on the work described in articles [54, 116, 117]. Unlike the previous FFNN-controllers which were trained using the original controller output as targets, this network will be trained using dynamic programming (DP). The DP algorithm incorporates conventional control techniques such as PID and predictive control, which gives advantages such as zero steady-state error, formulation of physical system constraints, and adaptive control behavior, even though the RNN is trained entirely offline [54].

This chapter is structured as follows: First, in Section 5.1 the analytical control model is described. Second, in Section 5.2 the mathematical model is given. In Section 5.3 the structure of the RNN is described. Section 5.4 focus on the forward accumulation through time (FATT) algorithm and yields the last design step.

There was not enough time to implement the controller, therefore there will be no evaluation of the controller performance. All scripts are given in the appendix D and includes extensive information about each step. This way it should be possible to continue where this implementation is stopped.

5.1 Analytical Model

The analytical control model of the proposed controller is given by Figure 5.1. The RNN-based controller is implemented in the synchronous reference frame, and replaces the PI controller once training is finished. The network takes as input the error signals $\boldsymbol{\varepsilon}_{dq} = i^*_{dq} - i_{c,dq}$ and the integrated errors $\boldsymbol{s}_{dq} = \int \varepsilon_{dq} dt$. The output of the RNN is equal to the outputs of the PI controller, namely the voltage signals $v^*_{c,dq}$. These control signals are transformed to *abc*-components by use of inverse-Park transform, then they are normalized with a gain equal to $\frac{2}{V_{dc}}$ before being sent to the sinusoidal pulse-width modulator.

5.2 Dynamic Program Development

The RNN will be trained to approximate an optimal controller using discrete-time DP. The discretized mathematical model of the system will be incorporated into the program and used to determine the optimal control signals. Therefore, the mathematical model is the starting point for the control design. Since the controller is implemented in the synchronous rotating reference frame, the system equations must be transformed to *dq*-components. This was previously done in Section 3.1.1 and the resulting mathematical model in state-space form is given below by Equation 5.1, here the subscript c refers to a continuous model. The model is discretized using *exact discretization*, as was described in Section 4.2.2, this results in the model defined by Equation 5.2.

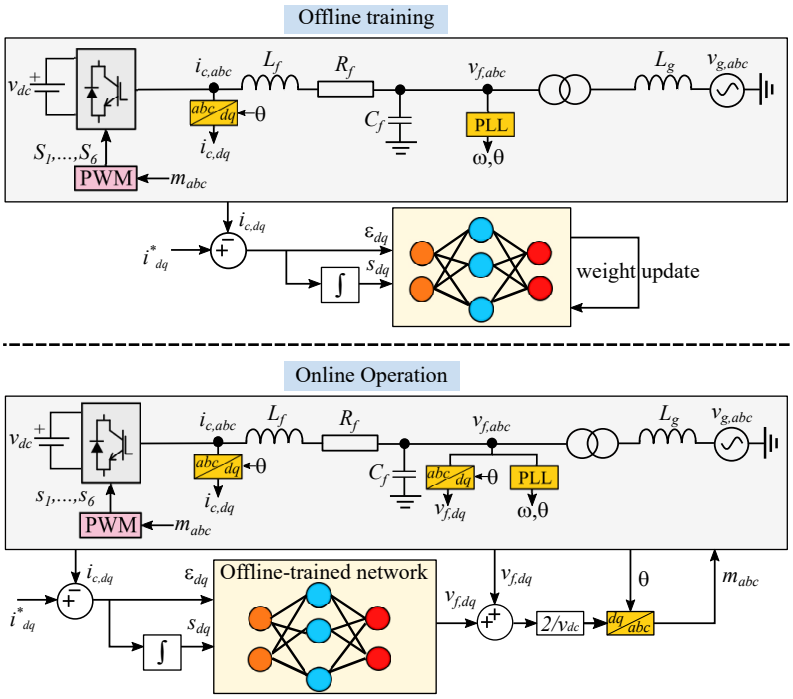


Figure 5.1: Schematic of the implementation of the RNN-based controller.

$$\underbrace{\frac{d}{dt}\begin{bmatrix} i_d \\ i_q \end{bmatrix}}_{\boldsymbol{i}_{dq}} = \underbrace{\begin{bmatrix} -\frac{R_f}{L_f} & \omega \\ -\omega & -\frac{R_f}{L_f} \end{bmatrix}}_{\mathbf{A}_c} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{1}{L_f} & 0 \\ 0 & \frac{1}{L_f} \end{bmatrix}}_{\mathbf{B}_c} \underbrace{\begin{bmatrix} v_{c,d} - v_{f,d} \\ v_{c,q} - v_{f,q} \end{bmatrix}}_{\boldsymbol{u}_{dq}} \tag{5.1}$$

$$\boldsymbol{i}_{dq}(k+1) = \underbrace{\begin{bmatrix} 1 - T_s\frac{R_f}{L_f} & T_s\omega \\ -T_s\omega & 1 - T_s\frac{R_f}{L_f} \end{bmatrix}}_{\mathbf{A}} \boldsymbol{i}_{dq}(k) + \underbrace{\begin{bmatrix} \frac{T_s}{L_f} & 0 \\ 0 & \frac{T_s}{L_f} \end{bmatrix}}_{\mathbf{B}} \boldsymbol{u}_{dq}(k) \tag{5.2}$$

The idea behind employing DP for training the RNN is to approximate an optimal controller according to the Bellman's optimality principle (see Subsection 2.4.1). Using Bellman's equation a cost function C can be defined. For this controller the cost function at time step k is defined as the sum of the squared errors in d and q-axis converter current, as given by Equation 5.3 for a discount factor γ, which is a positive, nonzero value below 1, a utility function U and a nonzero positive constant α. Here the objective is to choose a control signal $u_{dq}(k)$ such that C is minimized.

$$\begin{aligned} \min_{\boldsymbol{u}_{dq}(k)} C(\boldsymbol{i}_{dq}(k)) &= \sum_{j=k}^{\infty} \gamma^{j-k} U(\boldsymbol{\varepsilon}_{dq}(k)) \\ &= \sum_{j=t}^{\infty} \gamma^{j-t} \left(\varepsilon_d(k)^2 + \varepsilon_q(k)^2\right)^{\alpha} \\ &= \sum_{j=t}^{\infty} \gamma^{j-t} \left((i_d(k) - i_d^*(k))^2 + (i_q(k) - i_q^*(k))^2\right)^{\alpha} \end{aligned} \tag{5.3}$$

Since the system model is linear this minimization problem can be solved directly. The resulting control signals $\boldsymbol{u}_{dq}$ and converter voltages $\boldsymbol{v}_{c,dq}$ are defined by Equations 5.4 and 5.5, respectively.

$$\boldsymbol{u}_{dq}(k) = \mathbf{B}^{-1}\left(\boldsymbol{i}_{dq}^*(k+1) - \mathbf{A}\boldsymbol{i}_{dq}(k)\right) \tag{5.4}$$

$$\boldsymbol{v}_{f,dq}(k) = \mathbf{B}^{-1}\left(\boldsymbol{i}_{dq}^*(k+1) - \mathbf{A}\boldsymbol{i}_{dq}(k)\right) + \boldsymbol{v}_{f,dq} \tag{5.5}$$

The matrix $\mathbf{B}^{-1}(\mathbf{I}-\mathbf{A})$ is called a stabilization matrix. This matrix is obtained from Equation 5.4 for the case when $\boldsymbol{i}^*_{dq}(k+1)$ equals $\boldsymbol{i}_{dq}(k)$ as shown by Equation 5.6. This matrix defines the network output in terms of the network inputs and is considered as a recurrent connection, since now the information is flowing cyclic through the network. The motivation of using this stabilization matrix is to give the controller a default behaviour of holding the steady state and simplifies the neural network training. [116]

$$\boldsymbol{u}_{dq}(k) = \mathbf{B}^{-1}\left(\mathbf{I}-\mathbf{A}\right)\boldsymbol{i}_{dq}(k) \tag{5.6}$$

5.3 Recurrent Neural Network Architecture

The network structure is chosen equal to that proposed in [54] and is depicted in Figure 5.2. The network takes the two d- and q-axis error terms $\boldsymbol{\varepsilon_{dq}}$ defined by Equation 5.7, and the integral terms $\boldsymbol{s}_{dq}$ as inputs. The integral terms are evaluated using the trapezoid formula, for an initial error $\boldsymbol{\varepsilon}^\top_{dq}(0) = \begin{bmatrix}0 & 0\end{bmatrix}$, this is given by Equation 5.8. According to [54] this choice of inputs improves the network structure compared to the use of the currents $\boldsymbol{i}_{dq}$ and $\boldsymbol{i}^*_{dq}$ and their integrals as inputs. This is because the number of inputs is now reduced which limits the number of network weights and reduces the calculation effort during the online training.

$$\boldsymbol{\varepsilon}_{dq}(k) = \boldsymbol{i}_{dq}(k) - \boldsymbol{i}^*_{dq}(k) \tag{5.7}$$

$$\boldsymbol{s}_{dq}(k) = \int_0^{kT_s} \boldsymbol{\varepsilon}(t)dt \approx \frac{T_s}{2}\sum_{j=1}^{k} \boldsymbol{\varepsilon}_{dq}(j-1) + \boldsymbol{\varepsilon}_{dq}(j) \tag{5.8}$$

The network outputs the two control signals $\boldsymbol{v}_{c,dq}$ and has two hidden layers where each consists of 6 neurons. This gives the structure 4-6-6-2. The inputs are sent to the hyperbolic tangent function such that the values are normalized into the range $[-1,1]$, this is to avoid network input saturation. The hyperbolic tangent function is also used for all connections from the input to the hidden layer, and from the hidden layer to the output. Using the neural network relations from Section 2.3 the control signal $\boldsymbol{u}_{dq}$ can be defined according to Equation 5.9 for an input vector $\boldsymbol{p}^\top(k) = \begin{bmatrix}\varepsilon_d(k) & \varepsilon_q(k) & s_d(k) & s_q(k)\end{bmatrix}$. The network function $R(\boldsymbol{\varepsilon}_{dq}(k), \boldsymbol{s}_{dq}(k), \boldsymbol{w})$ is defined by Equation 5.9 and represents the operations of all network layers.

$$\begin{aligned}\boldsymbol{u}_{dq}(k) &= \boldsymbol{v}_{c,dq}(k) - \boldsymbol{v}_{f,dq}(k)\\ &= K_{PWM}R(\boldsymbol{\varepsilon}_{dq}(k), \boldsymbol{s}_{dq}(k), \boldsymbol{w}) - \boldsymbol{v}_{f,dq}(k)\end{aligned} \tag{5.9}$$

$$R(\boldsymbol{\varepsilon}_{dq}(k), \boldsymbol{s}_{dq}(k), \boldsymbol{w}) = \tanh\left(\mathbf{W}^3\tanh\left(\mathbf{W^2}\tanh\left(\mathbf{W}^1\boldsymbol{p}(k)\right)\right)\right) \tag{5.10}$$

5.4 Network Training

The RNN is trained to approximate an optimal controller. Therefore, unlike the two previous FFNN-based controllers in Chapters 3 and 4, this network will not use the original controller's outputs as targets. For this network the targets are generated through a training algorithm which is developed using principles from DP. This algorithm is known as LM plus FATT algorithm and will be described in the following subsections.

5.4.1 Levenberg-Marquardt

The LM algorithm, due to its speed and convergence guarantee, is one of the most widely used training algorithms for training FFNNs. The LM algorithm aims to minimize the cost defined by

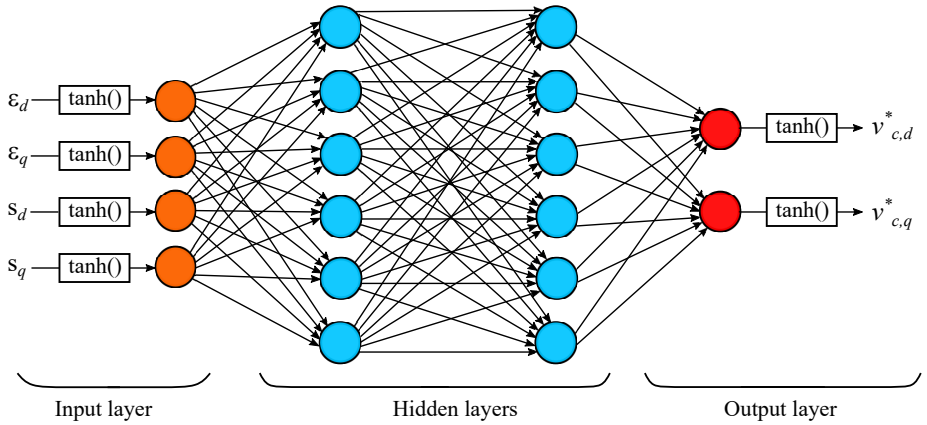


Figure 5.2: Recurrent neural network structure

Equation 5.11. Where $\boldsymbol{w}$ is a vector containing the elements of all network weights, $\boldsymbol{v}$ is a vector containing the errors of the network outputs for all samples N, since there are two outputs this results in $2N$ errors in total.

$$C(\boldsymbol{w}) = \sum_{i=1}^{2N} \boldsymbol{v}_i^2 \tag{5.11}$$

The weight update is defined by Equations 5.12 and 5.13 where the Jacobian matrix $\mathbf{J}$ is defined by Equation 5.14 for a total of M weight elements and N time steps. The calculation of the Jacobian elements, i.e. $\frac{\partial \boldsymbol{v}_j}{\partial w_i}$ will be described in the Section 5.4.2 being one of the core features of the FATT algorithm.

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + \Delta \boldsymbol{w} \tag{5.12}$$

$$\Delta \boldsymbol{w} = -\left(\mathbf{J}^\top \mathbf{J} + \mu \mathbf{I}\right)^{-1} \mathbf{J}^\top \boldsymbol{v} \tag{5.13}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \boldsymbol{v}_1}{\partial w_1} & \cdots & \frac{\partial \boldsymbol{v}_1}{\partial w_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{v}_N}{\partial w_1} & \cdots & \frac{\partial \boldsymbol{v}_N}{\partial w_M} \end{bmatrix} \tag{5.14}$$

If the $C(\boldsymbol{w})$ function is not a sum of squares the weight update in Equation 5.13 will not be directly applicable. Instead the cost defined by Equation 5.3 is used with a fractional α-value, as this improves the convergence during training according to [54]. This modification is illustrated by Equation 5.15. The gradient $\frac{\partial C}{\partial w}$ then becomes as defined in Equation 5.16.

$$C(\boldsymbol{i}_{dq}, \boldsymbol{w}) = \sum_{j=1}^{N} U(\boldsymbol{e}_{dq}(j)) \doteq \sum_{j=1}^{N} \boldsymbol{v}_j^2 \tag{5.15}$$

$$\frac{\partial C}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \sum_{j=1}^{N} \boldsymbol{v}_j^2 = 2 \sum_{j=1}^{N} \boldsymbol{v}_j \frac{\partial \boldsymbol{v}_j}{\partial \boldsymbol{w}} = 2\mathbf{J}^\top \boldsymbol{v} \tag{5.16}$$

5.4.2 Forward Accumulation Through Time

From Equation 5.16 it is obvious that the Jacobian matrix $\mathbf{J}$ is an essential part of the RNN training. According to FATT algorithm the k^{th} row of $\mathbf{J}$ is derived in Equation 5.17.

$$\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{w}} = \left[\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{\varepsilon}_{dq}(k)}\right] \left[\frac{\partial \boldsymbol{\varepsilon}_{dq}(k)}{\partial \boldsymbol{w}}\right] \tag{5.17a}$$

$$\frac{\partial \boldsymbol{v}_k}{\partial \boldsymbol{\varepsilon}_{dq}(k)} = \frac{\partial}{\partial \boldsymbol{\varepsilon}_{dq}(k)} \left(\varepsilon_d^2 + \varepsilon_q^2\right)^{\frac{\alpha}{2}} = \alpha(\varepsilon_d^2(k) + \varepsilon_q^2(k))^{\frac{\alpha}{2}-1} \begin{bmatrix} \varepsilon_d(k) & \varepsilon_q(k) \end{bmatrix} \tag{5.17b}$$

$$\frac{\partial \boldsymbol{\varepsilon}_{dq}(k)}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \left(\boldsymbol{i}_{dq}(k) - \boldsymbol{i}_{dq}^*(k)\right) = \frac{\partial \boldsymbol{i}_{dq}(k)}{\partial \boldsymbol{w}} \tag{5.17c}$$

The derivative $\frac{\partial \boldsymbol{i}_{dq}}{\partial \boldsymbol{w}}$ is computed using the discrete model defined in Equation 5.1, this results in Equation 5.18, where the initial derivative $\frac{\partial \boldsymbol{i}_{dq}}{\partial \boldsymbol{w}}(0)$ is equal to zero.

$$\frac{\partial \boldsymbol{i}_{dq}(k+1)}{\partial \boldsymbol{w}} = \mathbf{A}\frac{\partial \boldsymbol{i}_{dq}(k)}{\partial \boldsymbol{w}} + \mathbf{B}\frac{\partial \boldsymbol{u}_{dq}(k)}{\partial \boldsymbol{w}} \tag{5.18}$$

Next, $\frac{\partial \boldsymbol{u}_{dq}}{\partial \boldsymbol{w}}$ is computed using the network function $R(k) = R(\boldsymbol{\varepsilon}_{dq}(k), \boldsymbol{s}_{dq}(k), \boldsymbol{w})$ previously defined in Equation 5.10 and the relation $\frac{\partial \boldsymbol{\varepsilon}_{dq}}{\partial \boldsymbol{w}} = \frac{\partial \boldsymbol{i}_{dq}}{\partial \boldsymbol{w}}$ in Equation 5.17c. This results in Equation 5.19.

$$\frac{\partial \boldsymbol{u}_{dq}(k)}{\partial \boldsymbol{w}} = K_{PWM}\left[\frac{\partial R(k)}{\partial \boldsymbol{\varepsilon}_{dq}(k)}\frac{\partial \boldsymbol{i}_{dq}(k)}{\partial \boldsymbol{w}} + \frac{\partial R(k)}{\partial \boldsymbol{s}_{dq}(k)}\frac{\partial \boldsymbol{s}_{dq}(k)}{\partial \boldsymbol{w}} + \frac{\partial R(k)}{\partial \boldsymbol{w}}\right] \tag{5.19}$$

Finally, the derivative $\frac{\partial \boldsymbol{s}_{dq}}{\partial \boldsymbol{w}}$ is computed using the definition of the integral in Equation 5.8, this results in Equation 5.20.

$$\begin{aligned} \frac{\partial \boldsymbol{s}_{dq}(k)}{\partial \boldsymbol{w}} &= \frac{T_s}{2}\sum_{j=1}^{k}\frac{\partial}{\partial \boldsymbol{w}}\left(\boldsymbol{\varepsilon}_{dq}(j-1) + \boldsymbol{\varepsilon}_{dq}(j)\right) \\ &= T_s\left(\left(\sum_{j=0}^{k}\frac{\partial \boldsymbol{i}_{dq}(j)}{\partial \boldsymbol{w}}\right) - \frac{1}{2}\frac{\partial \boldsymbol{i}_{dq}(k)}{\partial \boldsymbol{w}}\right) \end{aligned} \tag{5.20}$$

Exploding Gradients

The exploding gradients problem refers to the accumulation of large error gradients which in turn makes the updates of the network weights become substantially large [118]. This problem can be better understood by analyzing the control system and network training. First, each time an element in the weight vector $\boldsymbol{w}$ is updated, the network actions chosen by $R(k)$ change at every time step. Each modified action $\Delta\boldsymbol{u}(k)$ consequently changes the next state $\Delta\boldsymbol{x}(k+1)$ the system passes through. Further, each modified state $\Delta\boldsymbol{x}(k+1)$ changes the next action chosen $\Delta\boldsymbol{u}(k+1)$ by Equation 5.10. This creates a cascade of changes which continues until training is finished. Hence, one small modification of an element in $\boldsymbol{w}$ can pose significant impacts on the trajectory developed by Equations 5.1 and 5.10. This is the major reason why training RNNs is hard. [116]

5.4.3 Forward Accumulation Through Time plus Levenberg-Marquardt

The complete FATT algorithm is given by Algorithm 1, here the process of calculating the Jacobian matrix $\mathbf{J}$ and the cost function C is given. Here $\mathbf{0}_{n\times m}$ defines a matrix of n rows and m columns, with all elements equal to zero, $\boldsymbol{\varepsilon}_{dq}(1,k)$ equals $\varepsilon_d(k)$ and $\boldsymbol{\varepsilon}_{dq}(2,k)$ equals $\varepsilon_q(k)$. The algorithm starts with initialization of the variables, next the network output is computed, after this in lines 4-6 the derivatives are computed according to Equations 5.20,5.20 and 5.18. The sum $\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{w}}$ in line 7 is necessary for computing the derivative of the integral term as given by Equation 5.20. In lines 8-10 the system trajectory is unrolled. Based on the system trajectory the utility function U and consequently the cost-to-go C are computed in lines 11-12. Next the derivatives necessary for computing the error vector $\boldsymbol{v}$ are calculated in lines 13-15. Finally the k^{th} row of the Jacobian $\mathbf{J}$ is obtained. These steps continue until all samples N have been processed.

Algorithm 1 Forward Accumulation Through Time algorithm

1: $C \leftarrow 0$, $\boldsymbol{\varepsilon}_{dq}(0) \leftarrow \mathbf{0}_{2\times 1}$, $\boldsymbol{s}_{dq}(0) \leftarrow \mathbf{0}_{2\times 1}$, $\frac{\partial \boldsymbol{i}_{dq}}{\partial \boldsymbol{w}} \leftarrow \mathbf{0}_{2\times M}$, $\frac{\partial \boldsymbol{\phi}_{dq}}{\partial \boldsymbol{w}} \leftarrow \mathbf{0}_{2\times M}$ ▷ Initialization
2: **for** $k = 0$ to $N-1$ **do** ▷ Calculate Jacobian $\mathbf{J}$
3: $\quad \boldsymbol{u}_{dq}(k) \leftarrow K_{PWM} R(k) - \boldsymbol{v}_{f,dq}$
4: $\quad \frac{\partial \boldsymbol{s}_{dq}(k)}{\partial \boldsymbol{w}} \leftarrow T_s \left[\frac{\partial \boldsymbol{\phi}(k)}{\partial \boldsymbol{w}} - \frac{1}{2} \frac{\partial \boldsymbol{i}_{dq}(k)}{\partial \boldsymbol{w}} \right]$
5: $\quad \frac{\partial \boldsymbol{u}_{dq}(k)}{\partial \boldsymbol{w}} \leftarrow K_{pwm} \left[\frac{\partial R(k)}{\partial \boldsymbol{\varepsilon}_{dq}(k)} \frac{\partial \boldsymbol{i}_{dq}(k)}{\partial \boldsymbol{w}} + \frac{\partial R(k)}{\partial \boldsymbol{s}_{dq}(k)} \frac{\partial \boldsymbol{s}_{dq}(k)}{\partial \boldsymbol{w}} + \frac{\partial R(k)}{\partial \boldsymbol{w}} \right]$
6: $\quad \frac{\partial \boldsymbol{i}_{dq}(k+1)}{\partial \boldsymbol{w}} \leftarrow \mathbf{A} \frac{\partial \boldsymbol{i}_{dq}(k)}{\partial \boldsymbol{w}} + \mathbf{B} \frac{\partial \boldsymbol{u}_{dq}(k)}{\partial \boldsymbol{w}}$
7: $\quad \frac{\partial \boldsymbol{\phi}(k+1)}{\partial \boldsymbol{w}} \leftarrow \frac{\partial \boldsymbol{\phi}(k)}{\partial \boldsymbol{w}} + \frac{\partial \boldsymbol{i}_{dq}(k+1)}{\partial \boldsymbol{w}}$
8: $\quad \boldsymbol{i}_{dq}(k+1) \leftarrow \mathbf{A} \boldsymbol{i}_{dq}(k) + \mathbf{B} \boldsymbol{u}_{dq}(k)$
9: $\quad \boldsymbol{\varepsilon}_{dq}(k+1) \leftarrow \boldsymbol{i}_{dq}(k+1) - \boldsymbol{i}^*_{dq}(k+1)$
10: $\quad \boldsymbol{s}_{dq}(k+1) \leftarrow \boldsymbol{s}_{dq}(k) + \frac{T_s}{2} (\boldsymbol{\varepsilon}_{dq}(k+1) + \boldsymbol{\varepsilon}_{dq}(k))$
11: $\quad U(k+1) \leftarrow (\varepsilon_d(k+1)^2 + \varepsilon_q(k+1)^2)^{\alpha}$
12: $\quad C \leftarrow C + U(k+1)$
13: $\quad \frac{\partial \boldsymbol{v}(k+1)}{\partial \boldsymbol{\varepsilon}_{dq}(k+1)} \leftarrow \alpha \left[\boldsymbol{\varepsilon}^2_{dq}(1, k+1) + \boldsymbol{\varepsilon}^2_{dq}(2, k+1) \right]^{\frac{\alpha}{2}-1} \left[\boldsymbol{\varepsilon}_{dq}(k+1) \right]^{\top}$
14: $\quad \frac{\partial \boldsymbol{\varepsilon}_{dq}(k+1)}{\partial \boldsymbol{w}} \leftarrow \frac{\partial \boldsymbol{i}_{dq}(k+1)}{\partial \boldsymbol{w}}$
15: $\quad \frac{\partial \boldsymbol{v}(k+1)}{\partial \boldsymbol{w}} \leftarrow \left[\frac{\partial \boldsymbol{v}(k+1)}{\partial \boldsymbol{\varepsilon}_{dq}(k+1)} \right] \left[\frac{\partial \boldsymbol{\varepsilon}_{dq}(k+1)}{\partial \boldsymbol{w}} \right]$
16: $\quad \mathbf{J}(k) \leftarrow \frac{\partial \boldsymbol{v}(k+1)}{\partial \boldsymbol{w}}$
17: **end for**

The full FATT+LM algorithm is illustrated in Figure 5.3. Here Algorithm 1 is incorporated into the fourth block named 'FATT'. The block denoted $FATT(')$ runs a modified version of Algorithm 1 which only computes the new cost C, based on the running of lines 3, 8, 9 and 12. The other blocks constitute the LM-algorithm. As was previously seen in the original LM algorithm in Figure 3.4, also here three stopping criteria are defined; a maximum learning rate μmax, a minimum norm of the gradient $||\frac{\partial C}{\partial \boldsymbol{w}}||$ and a maximum number of epochs $Epoch^{max}$, this is illustrated by the three yellow blocks which are all connected to the 'stop'-block, which will stop the network training.

In Appendix D the complete script for training the RNN is given, this includes preparing the inputs $\boldsymbol{\varepsilon}_{dq}$, $\boldsymbol{s}_{dq}$, computing all derivatives, and updating the weights. When running the script, the main issue is related to exploding gradient, which is a common problem for RNN training. There are several proposed strategies for fixing this issue given by [118–120], please refer to these for further improvement.

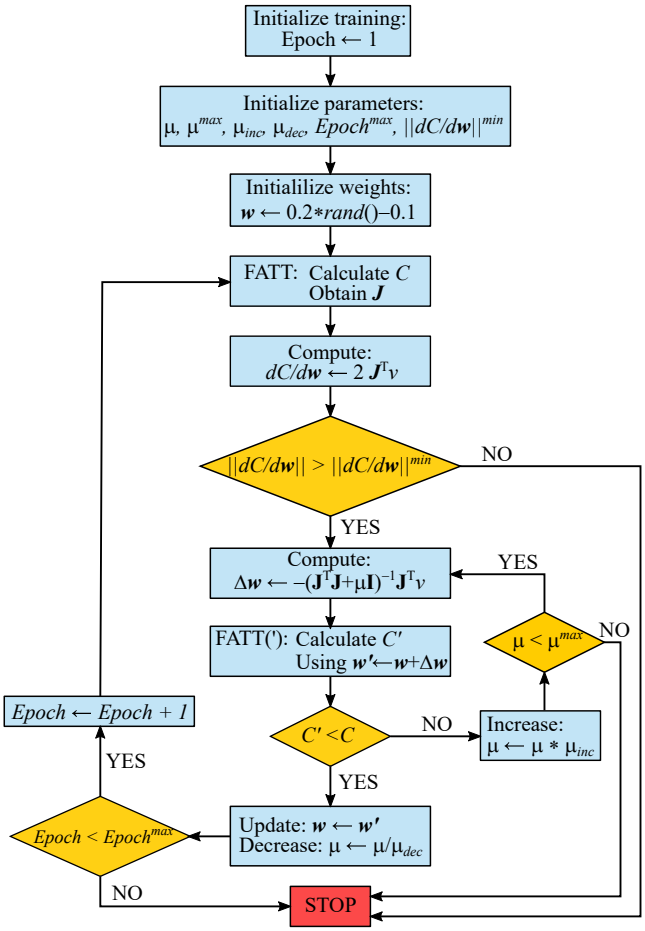


Figure 5.3: The full FATT+LM algorithm for training the recurrent neural network.

6

Direct Heuristic Dynamic Programming-Based Control

In this chapter an RL-based controller based on the direct heuristic dynamic programming algorithm is designed and implemented. The chapter is structured as follows. First, in Section 6.1 the control design is described, this is based on the work in [15]. Secondly, in Section 6.2 the control parameters are determined by analyzing the controller performance once the network is implemented. Third, in Section 6.3 the performance of the dHDP controller is evaluated. Finally, in Section 6.4 the simulation results are discussed and important remarks are made.

6.1 Control Design

The dHDP-method is an adaptive-critic design approach and is within reinforcement learning. The implementation procedure is illustrated in Figure 6.1. This controller consists of two neural networks which are pre-trained offline, using simulation data from the system, before implementation. During online operation the controller is implemented in parallel to the decoupled PI controllers and the weight updates continue. The errors $\varepsilon_{dq} = i^*_{dq} - i_{dq}$ are inputs to both the dHDP and the PI controllers. The output $\boldsymbol{u}_{dq}$ of the dHDP is added as a supplement to the output of the original PI controllers. The resulting control signals $v^*_{c,dq}$ are then transformed back into three phase components using inverse Park's transform, before being normalized, by multiplying with the gain $\frac{2}{V_{dc}}$. The modulation signals m_{abc} are sent to the PWM. These last steps are identical to what was described in Section 3.1.2 for the decoupled PI controller. The control block named Direct HDP represents the training algorithm which updates the weights of the two networks, called critic and actor, which together constitute the supplementary dHDP controller.

6.1.1 Direct HDP Principle

Direct HDP is a part of the family of adaptive DP methods, which all seek to find an approximately optimal control policy for a stochastic process whose model depends on unknown parameters [121]. In dHDP the control signals are estimated directly, unlike indirect versions which estimate model parameters before computing the control signals, and enables the use of a model-free controller which is robust with respect to model uncertainties [89, 96]. This is illustrated in Figure 6.2 which illustrates the working of the dHDP method. Here the orange arrows ($x(t)$) shows that the information from the process is directly used to find appropriate controller parameters.

In Figure 6.2 $\boldsymbol{x}(t)$ represents the inputs to the controller, which in this case consists of the two error terms in dq-frame $\boldsymbol{\varepsilon}_{dq}$ and $\boldsymbol{u}(t)$ represent the control signals $\boldsymbol{u}_{dq}$. The basic idea in an

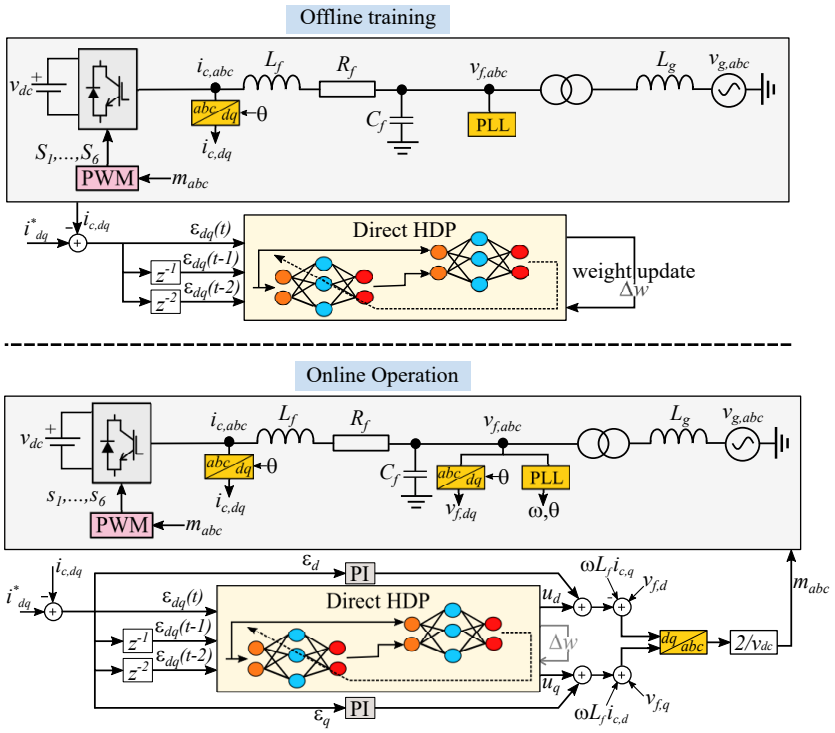


Figure 6.1: The implementation of the dHDP controller.

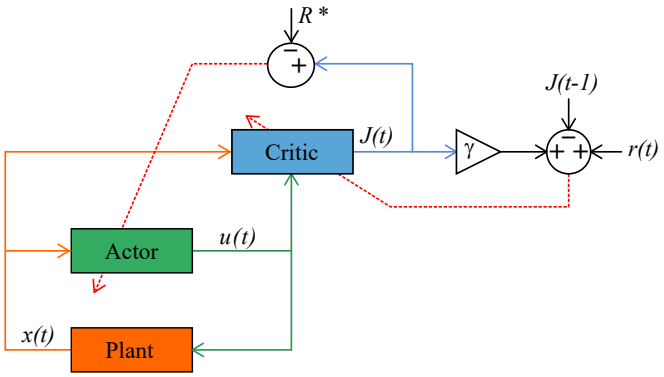


Figure 6.2: Principle of the direct heuristic dynamic programming control method.

adaptive-critic design is to adapt the weights of the critic network such that the approximate optimal cost function, $J(X(t))$, satisfies the modified Bellman equation. This is indicated by the red arrow, which illustrates that the critic aims to minimize the value $\gamma J(t) + r(t) - J(t-1)$, for a reinforcement signal $r(t)$ and a discount factor γ. The actor, also known as the action network, outputs the control signal $u(t)$ obtained by minimizing the value of $J(t) - R^*(t)$.

6.1.2 Cost-To-Go Function

The control design starts with defining J^* which describes the cost-to-go of inputs $\boldsymbol{x}(t)$. According to Bellman this can be expressed by Equation 6.1, where γ ($\in 0, 1$) is a discount rate and determines the relative importance of the present reward compared to future rewards. The reinforcement signal $r(t)$ incorporates the control objective for a particular scenario in one or more measurable variables.

$$J^*(t) = \sum_{i=0}^{\infty} \gamma^i r(t+i) \tag{6.1}$$

A direct solution to this optimization task proves to be computationally infeasible due to the curse of dimensionality (Section 2.4.1), therefore a more tractable approximation method is developed using Equation 6.2 [122]

$$J^*(t) = r(t) + \gamma J^*(t+1) \tag{6.2}$$

In this method, which is based on the article in [15] the critic network is, however, used to approximate the future cost-to-go, that is $J(t) = J^*(t+1)$, which is expressed by Equation 6.3

$$J(t) = J(t-1) - r(t) \tag{6.3}$$

This leads to an optimal control signal as given in Equations 6.4 and 6.5. Here R^* is the ultimate desired objective that the cost function aims to achieve and is a binary value. These equations are approximated in an iterative manner using two FFNNs, namely the actor and the critic. As their names imply the actor approximates the actions u and the critic evaluates the actor's performance by approximating J.

$$J[x(t)] = \min_{\boldsymbol{u}^*(t)} \{J[\boldsymbol{x}(t+1)] + r[\boldsymbol{x}(t)] - R^*\} \tag{6.4}$$

$$\boldsymbol{u}^*(x(t)) = arg \min_{\boldsymbol{u}^*(t)} J[\boldsymbol{x}(t)] \tag{6.5}$$

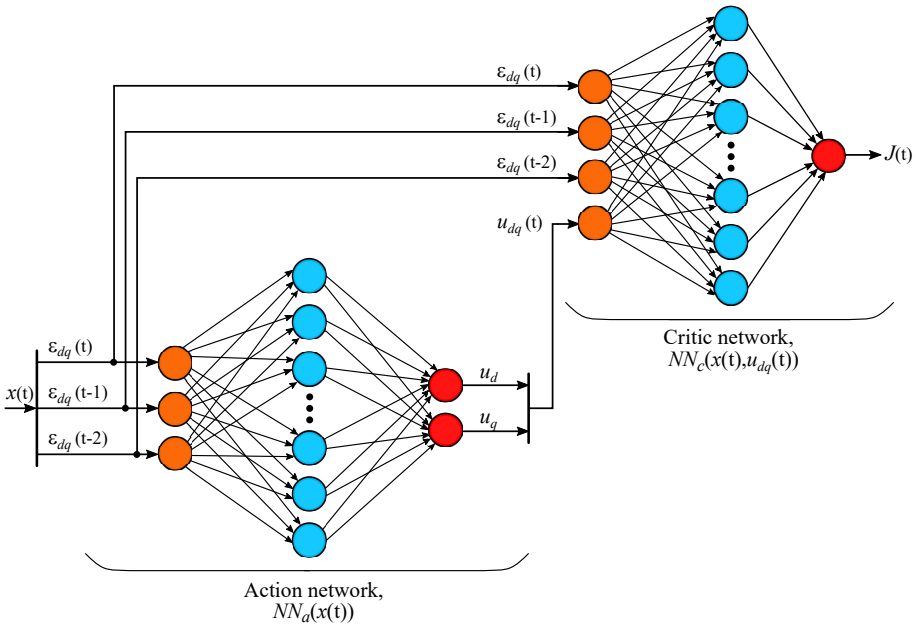


Figure 6.3: Network structure of critic and action networks.

6.1.3 Critic Network

As illustrated in Figure 6.2, the critic takes both the control signals $\boldsymbol{u}(t)$ generated by the actor and the measured system states $\boldsymbol{x}(t)$ as inputs, which in total becomes 8 inputs (u_{dq}, $\varepsilon_{dq}(t)$, $\varepsilon_{dq}(t-1)$, $\varepsilon_{dq}(t-2)$). As proposed in [15], one hidden layer with 12 neurons is chosen. The output of the critic is the approximated cost-to-go function J, this is described by Equation 6.6, where NN_c represents the operations done by the feedforward network and $\boldsymbol{w}_c$ is a vector containing all the weights of the network . In Figure 6.3 the network structures of the critic and actor are illustrated, where the critic takes the structure 8-12-1.

$$J(t) = NN_c(x(t), u(t), \boldsymbol{w}_c) \tag{6.6}$$

The aim of the critic is to approximate the value function, here referred to as the cost-to-go function J. This is done by minimizing the objective function E_c defined by Equation 6.7. The prediction error e_c reflects the difference in the current and previous cost-to-go functions $J(t)$ and $J(t-1)$,

in addition to the amount of reinforcement that is currently implemented $r(t)$, this is described by Equation 6.8. The goal is that J converges towards the optimal cost-to-go J^* while keeping the reinforcement signal at a minimum, only then an appropriate value function is obtained. As given by Equation 6.9 the reinforcement signal is determined by the amount of deviation between reference and measured currents, ε_{dq}, for the current and previous time steps. The deviation at previous time steps, $\varepsilon_{dq}(t-1)$ and $\varepsilon_{dq}(t-2)$, is included to limit the strength of the control signals during steady state, while allowing stronger control signals during transient conditions. The coefficients a_1, a_2 and a_3 are kept constant and are chosen equal to those proposed in [15].

$$E_c(t) = \frac{1}{2} e_c(t)^2 \tag{6.7}$$

$$e_c(t) = \gamma J(t) - J(t-1) + r(t) \tag{6.8}$$

$$r(t) = -(a_1 \varepsilon_{dq}^2(t) + a_2 \varepsilon_{dq}^2(t-1) + a_3 \varepsilon_{dq}^2(t-2)) \tag{6.9}$$

The weights $\boldsymbol{w}_c$ are updated according to gradient descent rule as given by Equation 6.10. Here the learning rate lr_c is a small and positive number, and unlike the learning rate in the LM algorithm it remains constant throughout the training period. The gradient $\frac{\partial E_c}{\partial \boldsymbol{w}_c}$ represents the change in the objective function based on the change in the weights and is computed using Equation 6.11

$$\boldsymbol{w}_{c,new} = \boldsymbol{w}_{c,old} + \Delta \boldsymbol{w}_c = \boldsymbol{w}_{c,old} - lr_c \frac{\partial E_c}{\partial \boldsymbol{w}_c} \tag{6.10}$$

$$\frac{\partial E_c}{\partial \boldsymbol{w}_c} = \left[\frac{\partial E_c}{\partial e_c}\right] \left[\frac{\partial e_c}{\partial J}\right] \left[\frac{\partial J}{\partial \boldsymbol{w}_c}\right] = e_c \gamma \frac{\partial J}{\partial \boldsymbol{w}_c} \tag{6.11}$$

6.1.4 Action Network

The network structure of the actor is also depicted in Figure 6.3. Similar to the critic network, the actor is also designed as an FFNN with 12 neurons in the hidden layer. The action network takes the measured states of the system, that is

$$\boldsymbol{x}(t) = \begin{bmatrix} \varepsilon_{dq}(t) \\ \varepsilon_{dq}(t-1) \\ \varepsilon_{dq}(t-2) \end{bmatrix}$$

as inputs, which results in 6 inputs. The network outputs the two control signals $\boldsymbol{u}_{dq}(t)$ as expressed by Equation 6.12, where NN_a represents all the network operations, and $\boldsymbol{w}_a$ is the weight vector of the action network.

$$\boldsymbol{u}_{dq} = NN_a(\boldsymbol{x}(t), \boldsymbol{w}_a) \tag{6.12}$$

The actor approximates the policy (Subsection 2.4.1) by selecting appropriate actions based on the cost-to-go J, this is given by Equation 6.13. Considering Equation 6.9 the ultimate objective is a reinforcement signal equal to zero, this leads to an ultimate objective $R^*(t) = 0$. Thus, in this design the action network prediction error is set equal to the value of the approximated cost-to-go function, $e_a(t) = J(t)$.

$$E_a(t) = \frac{1}{2} e_a^2(t) = \frac{1}{2} (J(t) - R^*(t))^2 \tag{6.13}$$

Similar to the critic network, the weights of the actor $\boldsymbol{w}_a$ are updated according to gradient descent, this is given by Equation 6.14, and the gradient $\frac{\partial E_a}{\partial w_a}$ is defined by Equation 6.15.

$$\boldsymbol{w}_{a,new} = \boldsymbol{w}_{a,old} + \Delta \boldsymbol{w}_a = \boldsymbol{w}_{a,old} - lr_a \frac{\partial E_a}{\partial \boldsymbol{w}_a} \tag{6.14}$$

$$\frac{\partial E_a}{\partial w_a} = \frac{\partial E_a}{\partial J} \frac{\partial J}{\partial u_{dq}} \frac{\partial u_{dq}}{\partial w_a} \tag{6.15}$$

6.1.5 Direct HDP Training Algorithm

Unlike the two previous FFNN-based control methods which employ techniques within SL, the dHDP controller use an RL-based training method. More specifically, the training of the critic and action networks are done in two stages, first offline and second online. For both stages the network weights are updated based on gradient descent, given by Equations 6.10 and 6.11 for the critic and 6.14 and 6.15 for the actor. For the offline training the weights are initialized randomly for both the critic and actor. For the online training, the weights obtained after the offline training are used as initial weights. Despite this, the training algorithm remains the same.

The first step of the algorithm is to initialize the weights for the critic $\boldsymbol{w}_c$ and actor $\boldsymbol{w}_a$, the control signal $\boldsymbol{u}_{dq}$ and the cost-to-go J as is shown in line 1 of Algorithm 2. The second step is to calculate the control signal $\boldsymbol{u}_{dq}$ using the action network and the cost-to-go using the critic network. Note that a gain G is included for the control action, this was proposed in [15] and is further discussed in Section 6.2. After implementing the control signals, the input at the next time step $\boldsymbol{x}(t+1)$ and the reinforcement signal $r(t)$ is obtained, as is seen in lines 3-5. Based on this data, in lines 6-11, the algorithm goes through an update loop for the critic and continues until a maximum number of iterations k_c^{max} is reached or until the objective function value falls below a minimum value E_c^{min}. Next, a similar update loop, in lines 13-19, is performed for the actor, this is stopped for similar criteria, a maximum number of iterations k_a^{max} and a minimum objective function value E_a^{min}. These two loops lead to updated weights for the critic and actor. In lines 20-21 the input vector is updated and the cost-to-go value is stored. The developed scripts for offline and online training based on the dHDP-algorithm is given in Appendix F.

Algorithm 2 Direct HDP training algorithm

1: initialize W_c, W_a, $u_{dq} \leftarrow \begin{bmatrix} 0 & 0 \end{bmatrix}^\top$, $J \leftarrow 0$, $k_a \leftarrow 1$, $k_c \leftarrow 1$
2: **repeat**
3: $\quad \boldsymbol{u}_{dq}(t) \leftarrow G \times NN_a(\boldsymbol{x}(t), \boldsymbol{w}_a)$
4: $\quad J(t) \leftarrow NN_c(\boldsymbol{x}(t), \boldsymbol{u}_{dq}(t), \boldsymbol{w}_c)$
5: $\quad$ insert $\boldsymbol{u}_{dq}$, get $\boldsymbol{x}(t+1)$, compute $r(t)$
6: $\quad$ **while** $E_c \geq E_c^{min}$ and $k_c \leq k_c^{max}$ **do**
7: $\quad\quad e_c(t) \leftarrow \gamma J(t) - J(t-1) + r(t)$
8: $\quad\quad E_c(t) \leftarrow \frac{1}{2} e_c^2(t)$
9: $\quad\quad \Delta W_c \leftarrow -l \frac{\partial E_c}{\partial W_c}$
10: $\quad\quad W_c \leftarrow W_c + \Delta W_c$
11: $\quad\quad k_c \leftarrow k_c + 1$
12: $\quad$ **end while**
13: $\quad$ **while** $E_a \geq E_a^{min}$ and $k_a \leq k_a^{max}$ **do**
14: $\quad\quad e_a(t) \leftarrow J(t)$
15: $\quad\quad E_a(t) \leftarrow \frac{1}{2} e_a^2(t)$
16: $\quad\quad \Delta W_a \leftarrow -l \frac{\partial E_a}{\partial W_a}$
17: $\quad\quad W_a \leftarrow W_a + \Delta W_a$
18: $\quad\quad k_a \leftarrow k_a + 1$
19: $\quad$ **end while**
20: $\quad \boldsymbol{x}(t) \leftarrow \boldsymbol{x}(t+1)$
21: $\quad J(t-1) \leftarrow J(t)$
22: **until** simulation is finished

6.2 Determining Control Parameters

In the dHDP-based control design there are several parameters which must be determined. All control parameters were initially chosen equal to those the authors in [15] propose, however some changes were found necessary and the finalized control parameter values are listed in Table 6.1. More specifically, the proposed values for the discount factor γ and the controller gain G did not give a satisfactory controller performance and had to be modified. To choose appropriate values

the critic and actor are fully trained both offline and online for different γ- and G-values. Then the controller performance is observed for a step from 5A to 20A at 0.1s in current reference for all values. The results are discussed in the following subsections.

Table 6.1: Direct HDP controller parameters

Symbol	Description	Value
	Critic network structure	8-12-1
	Action network structure	6-12-2
lr	Learning rate (offline, online)	0.01, 0.001
k_c^{max}	Iterations, critic (offline, online)	20, 50
k_a^{max}	Iterations, actor (offline, online)	10, 40
E_c^{min}	Minimum error for critic	10^{-6}
E_a^{min}	Minimum error for actor	10^{-7}
a_1, a_2, a_3	Reinforcement coefficients	0.4, 0.2, 0.04
γ	Discount factor	0.1
G	Controller gain	1

6.2.1 Discount Factor

The discount factor γ in Equation 6.8 was in [15] set equal to 0.95, however a too high γ resulted in oscillatory behaviour. The controller performance for three different γ values is presented in Figure 6.4. Here it is seen that the performance when using γ=0.7 is acceptable until approximately 0.17s, where large deviations are seen to occur. Further reducing γ to 0.5 resulted in a significant performance improvement, however some large deviations are still observed at approximately 0.12s and 0.25s. Using γ=0.1 removed all the large and abrupt deviations. In the bottom right corner a close-up of the step at 0.1s is plotted. Here it is seen that smaller γ results in an increased response speed. Thus, γ=0.1 resulted in the overall best performance and was therefore chosen.

6.2.2 Controller Gain

In [15] the authors propose to insert a gain G equal to 100 to the output of the dHDP controller. A number of different gains was tested and the best performance was obtained for $G = 1$. The tracking performance of the controller using this gain is presented in Figure 6.5. For this system an increased gain resulted in increased deviation, and slower dynamic response. The unity gain was therefore chosen.

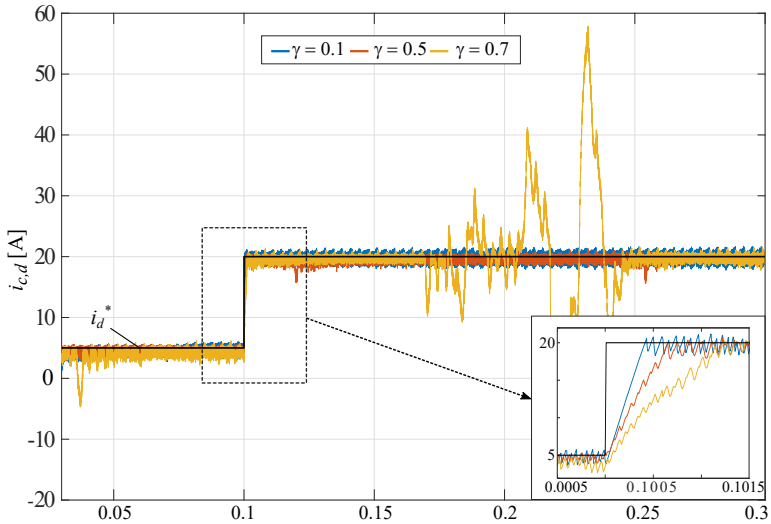


Figure 6.4: Impact of discount factor γ on the dHDP control performance.

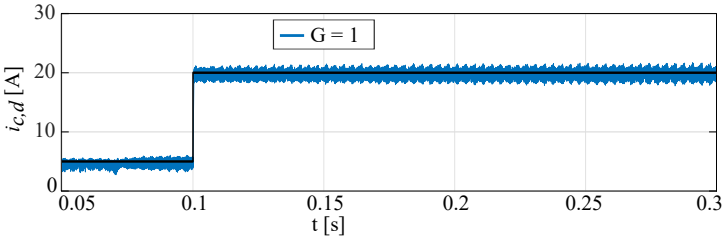


Figure 6.5: Impact of controller gain G on the dHDP control performance.

6.3 Performance Evaluation

In this section, the performance of the dHDP controller is evaluated. The controller is implemented in the Matlab/Simulink environment for a switching model of the VSC, using the system parameters previously given in Table 3.1 and the control parameters in Table 6.1. The evaluation is based on three different scenarios which will be described in each following subsection.

6.3.1 Current Reference Tracking

The tracking ability of the dHDP controller is evaluated by varying the d-axis reference current magnitude (i_d^*) continuously and step-wise. The obtained results are presented in Figure 6.6 and include plots of the d-axis converter current $i_{c,d}$, phase a of the filtered output current $i_{f,a}$ and phase a of the filtered output voltage $v_{f,a}$. It can be verified that the dHDP controller is successfully implemented, as the actual converter current ($i_{c,d}$) accurately tracks the corresponding reference component (i_d^*). The deviation between the actual current and the reference is reduced when the dHDP controller is implemented, compared to the original PI controller. In the close-up of the step at 0.12s in the top plot, the improved performance is further verified, as the response speed for the dHDP controller is significantly faster than the PI controller. The current $i_{f,a}$ has a smooth waveform for both controllers, however, slightly larger distortions are observed for the PI controller right after the reference current step is introduced. The voltage $v_{f,a}$ has a smooth waveform for both controllers.

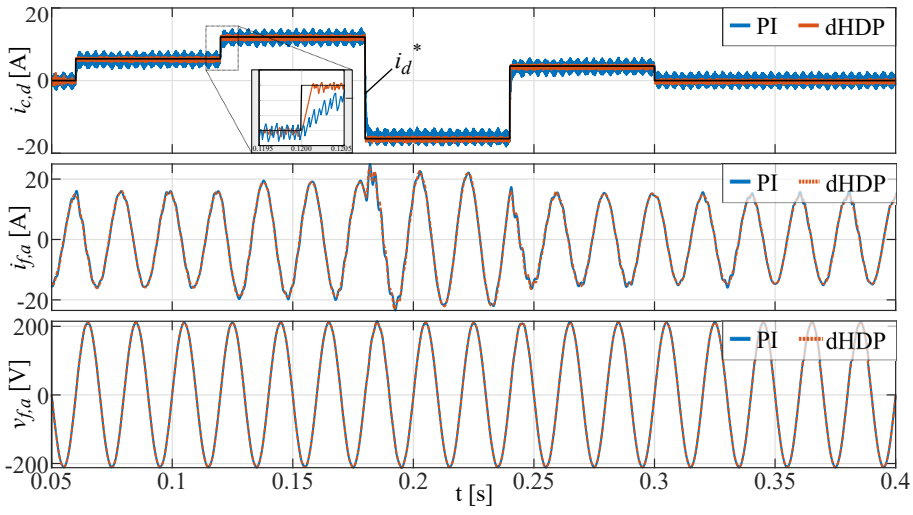


Figure 6.6: Direct HDP tracking performance for a step-varying reference current.

6.3.2 Analysis During Grid-Side Fault

In Figure 6.7 the performance of the dHDP-based controller and the PI controller is tested under grid-side fault conditions by reducing the grid voltage amplitude $|v_g|$ symmetrically to 20%. The short circuit fault is introduced at 0.2s and lasts for 3 cycles, while the reference current magnitude is kept constant at 18A. In Figure 6.7 the d-axis converter current $i_{c,d}$, phase-a of the filtered output current $i_{f,a}$ and filtered output voltage $v_{f,a}$ are presented. When the short circuit occurs the deviation between actual d-axis current and its corresponding reference decreases for both controllers, this lasts until the fault is cleared at 0.26s. At the moment of short circuit clearing the $i_{c,d}$-value has a large drop for both controllers, however the drop is larger when only the PI control is used. The voltage and current waveforms, $v_{f,a}$ and $i_{f,a}$, become momentarily distorted when the grid voltage drops, and rises. The level of distortions appear identical for the two controllers.

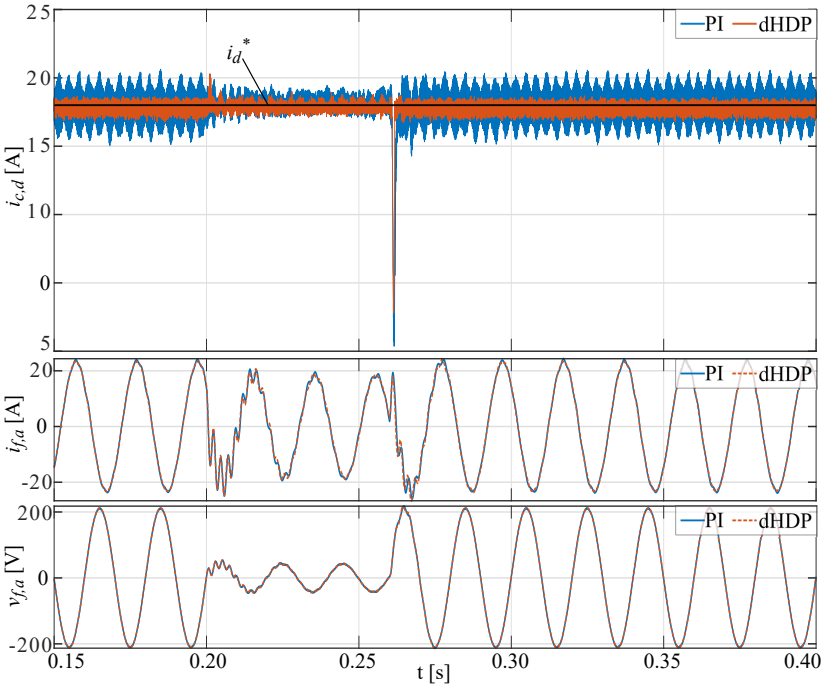


Figure 6.7: Direct HDP controller performance when a short circuit fault is introduced at the grid-side.

6.3.3 Parameter Uncertainty

In this test case the robustness of both controllers is verified for modified grid and filter inductance. The filter inductance L_f is increased and decreased by 20% and the grid inductance L_g is increased and decreased by 300%. The i_d^*-value is also changed from 5A to 20A at 0.2s.

In Figure 6.8 the $i_{c,d}$ and i_d^* are presented for both controllers. Here it is seen that both controllers successfully track the reference current. The oscillations in $i_{c,d}$ is lower for the dHDP for all parameter modifications. While the PI controller has an approximately constant level of $i_{c,d}$ oscillations, the oscillations vary with time for the dHDP controller. The dHDP controller also has a faster response for all cases, which is seen in the close-up in the bottom right corner.

The filtered output current $i_{f,abc}$ is presented in Figure 6.9 for the dHDP controller. Here it is seen that some distortions are present when the reference current steps up to 18A at 0.2s. When the grid inductance is increased it can be seen that it takes slightly more time before the distortions are cleared out. The distortions are greatest for phase a of the current as this is first affected by the step change in current reference. Despite this, the current waveform is smooth before and after the reference current step for all cases of parameter modifications.

In Figure 6.10 the filtered output voltage $v_{f,abc}$, when the dHDP controller is implemented, is presented for the same parameter modifications as above. The waveform is seen to be smooth for all cases, but has some distortions when the reference current steps up to 18A at 0.2s, this visibly affects phase a of the voltage and has some smaller impacts on phase b and c. These distortions are highest for the case when the grid inductance is increased. However for all cases the distortions are quickly cleared out.

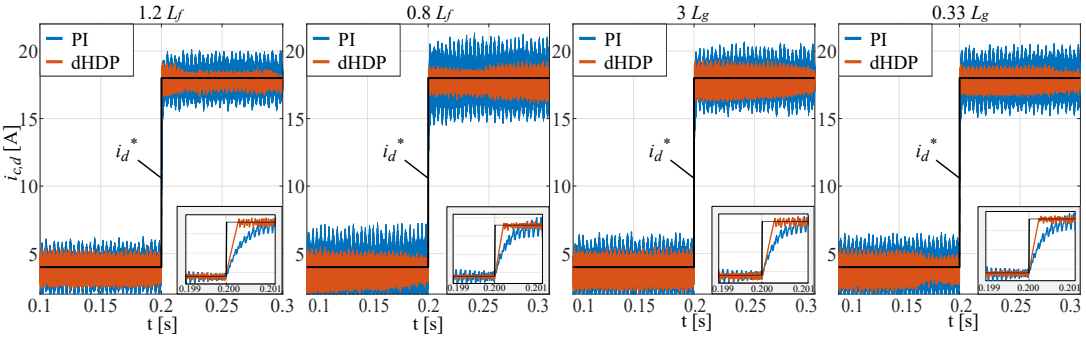


Figure 6.8: D-axis converter current for the parameter uncertainty test case.

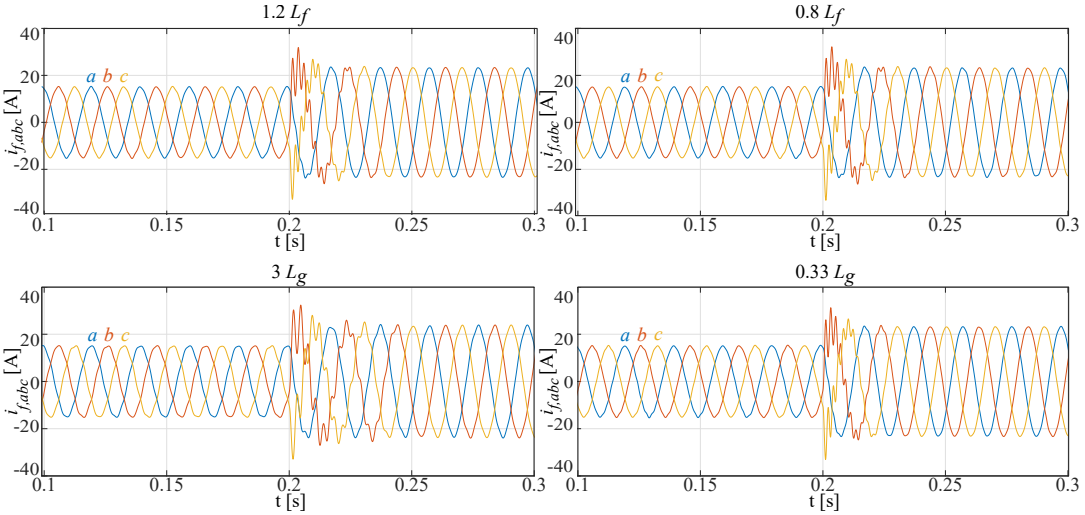


Figure 6.9: Filtered output current for the parameter uncertainty test case.

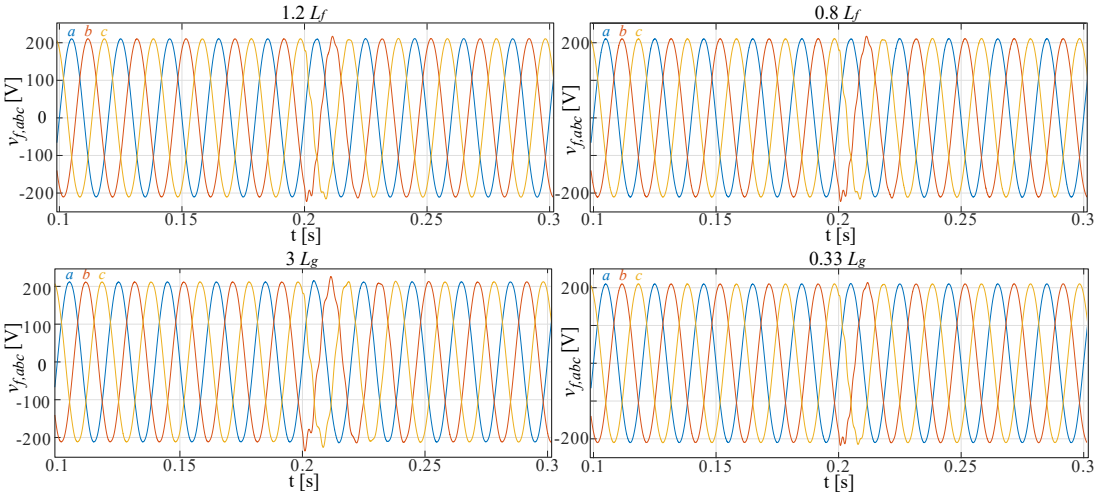


Figure 6.10: Filtered output voltage for the parameter uncertainty test case.

6.4 Discussion

For all test cases, the dHDP shows an improved performance compared to the conventional PI controller. In this thesis, results have only been obtained using the Matlab/Simulink environment, and a validation through HIL simulations is recommended. The dHDP controller consists of heavy computations, mainly due to the inclusion of two while-loops which must be executed in each time step. As a result, simulating this controller as currently designed, takes significant amount of time. Thus, changes to the design could improve the simulator performance. It is also worth mentioning that the simulation results were altered by the processing capacity of the computer on which the simulations were performed. This required restarting the computer occasionally and minimizing the number of active programs. Improvements in the simulation speed can be achieved by replacing the 'Matlab Function' block with the 'built-in Function' block, removing excess data logging and subsystems. For more information on improving the simulation speed please refer to [123]. It is also worth mentioning that there are numerous user defined parameters, and as was experienced during the network design phase, changing these could lead to an improved controller performance.

7

Comparison of Controller Performance

In this chapter a comparison of the performances of all the proposed control methods is presented. The same test cases as in the previous performance evaluations will be employed. This chapter is organized as follows. First, in Section 7.1 the implementation of all controllers is briefly described, including an analytical model of the system and controllers. Next, in Section 7.2 the performance of the controllers for the varying reference current test case is compared. In Section 7.3 the performance of the controllers is compared for the case when a short circuit fault is introduced at the grid-side. In Section 7.4 the performance is compared for the parameter uncertainty test case. Finally, in Section 7.5 all simulation results are discussed and concluding remarks are given.

This chapter is a continuation of the article 'Performance Evaluation of ANN-Based Control of a Grid-Connected Converter with Different Training Datasets'. The article was submitted for publication and is given in Appendix A.

7.1 Implementation

In Figure 7.1 an analytical model of the system and the proposed control methods is illustrated. The system consists of the grid-connected VSC, an LC filter and the distribution grid. In addition, the modulation technique, which consists of pulse-width modulation (PWM) and the grid synchronization technique, the phase-locked loop (PLL), are illustrated.

The decoupled PI controller is included in the following analysis. This control method represents the conventional control method and is included for comparison reasons. The ANN-based controllers consists of the two SL-based controllers, the PI-ANN and the MPC-ANN controllers, and the RL-based controller, the dHDP controller. The following analysis will compare the ANN-based controllers' performances to each other and to the conventional control method. An analytical model of the implementation of all controllers is presented by Figure 7.1. The figure shows the inputs and outputs to each ANN and the necessary component transformations. The outputs of all control methods are the modulation signals m_{abc}. The modulation signals are sent to the PWM which generates the appropriate switching signals S_1, S_2,..., S_6.

The system is implemented in the Matlab/Simulink simulation environment and all control methods are implemented for a switching model of the VSC considering the system parameters given by Table 3.1. The control parameters are the same as in previous chapters and are given by Table 3.3 for the PI and PI-ANN controllers, the parameters for the MPC-ANN controller is given by Table 4.1 and the dHDP control parameters are given by Table 6.1.

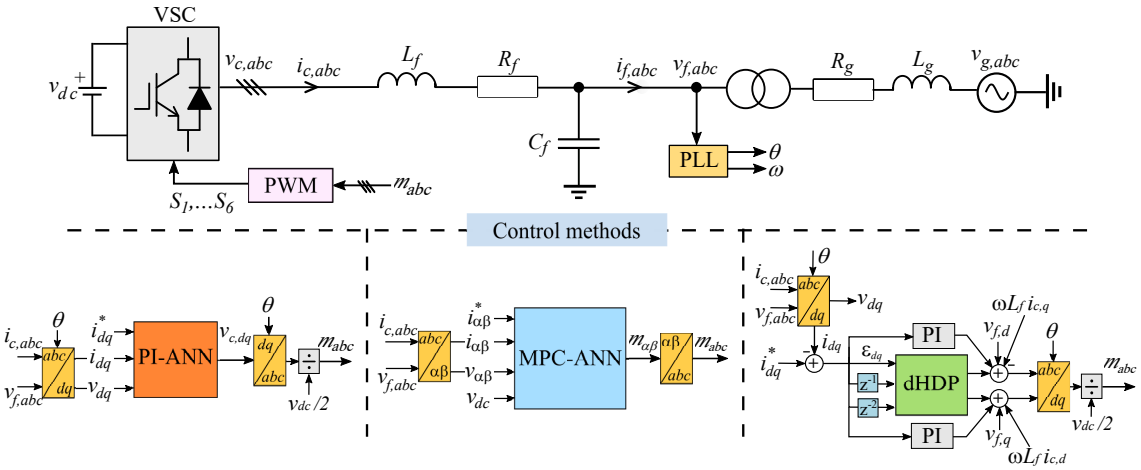


Figure 7.1: Implementation of proposed controllers.

7.2 Current Reference Tracking

A simulation is carried out for all the controllers by varying the d-axis reference current magnitude (i_d^*) continuously and step-wise. The resulting d-axis converter current $i_{c,d}$, phase-a of filtered output current $i_{f,a}$ and phase-a of filtered output voltage $v_{f,a}$ under this test case are presented in Figure 7.2.

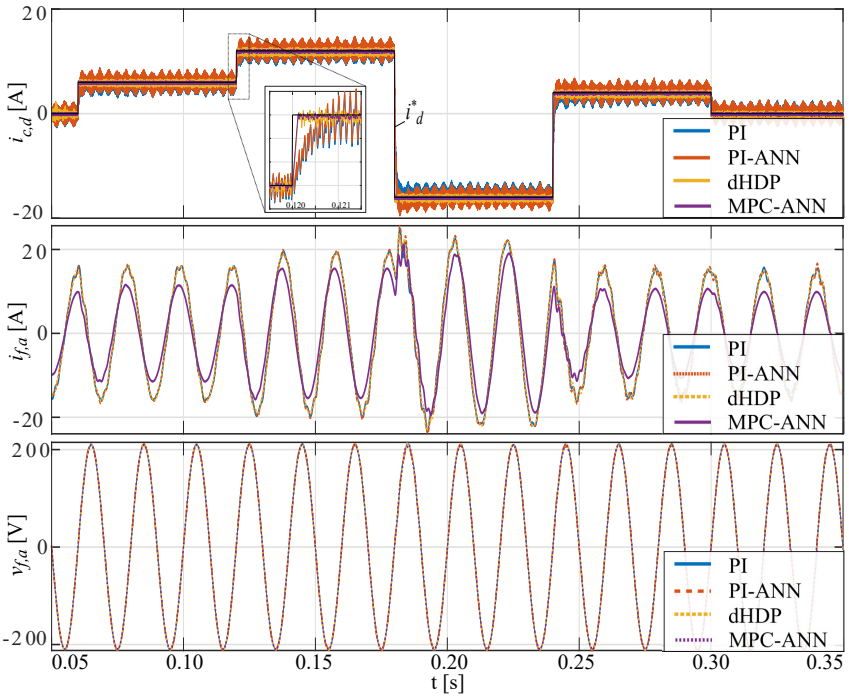


Figure 7.2: Performance evaluation of all controllers under reference current variation.

The successful implementation of all the controllers can be verified from Figure 7.2, as the measured d-axis converter current is accurately tracking the corresponding reference component. However the deviation between the measured current and its reference is seen to be larger for the PI and PI-ANN controllers. The dHDP controller has more accurate tracking. Further, from the close-up it is seen that the dHDP controller also exhibits a faster response speed. The MPC-ANN controller shows the best tracking performance, where the deviation is almost not visible in the chosen scaling. The response speed, however, appears identical to that of the dHDP-controller.

Further, the filtered output current $i_{f,a}$ appears identical for all controllers, except for the MPC-ANN controller. For this controller the $i_{f,a}$-magnitude sets at a different value. In addition, the waveform exhibits less distortion, than what is seen for the other controllers. However, looking closely, the dHDP-controller also appears less distorted than the PI and PI-ANN controllers. For all controllers it seen an increase in the $i_{f,a}$-distortions for the large reference current steps at 0.18s and 0.24s. The $v_{f,a}$ waveform appears identically smooth for all controllers, and is satisfactorily maintained at the desired level.

7.3 Analysis During Grid-Side Fault

The performance of all the controllers is also tested under grid-side fault conditions by reducing the grid voltage amplitude symmetrically to 20%. the fault is introduced at 0.2 s and lasts for 3 cycles, while the reference current is kept constant at 18A. The d-axis converter current, phase-a of the filtered output current and phase-a of the filtered output voltage is presented in Figure 7.3.

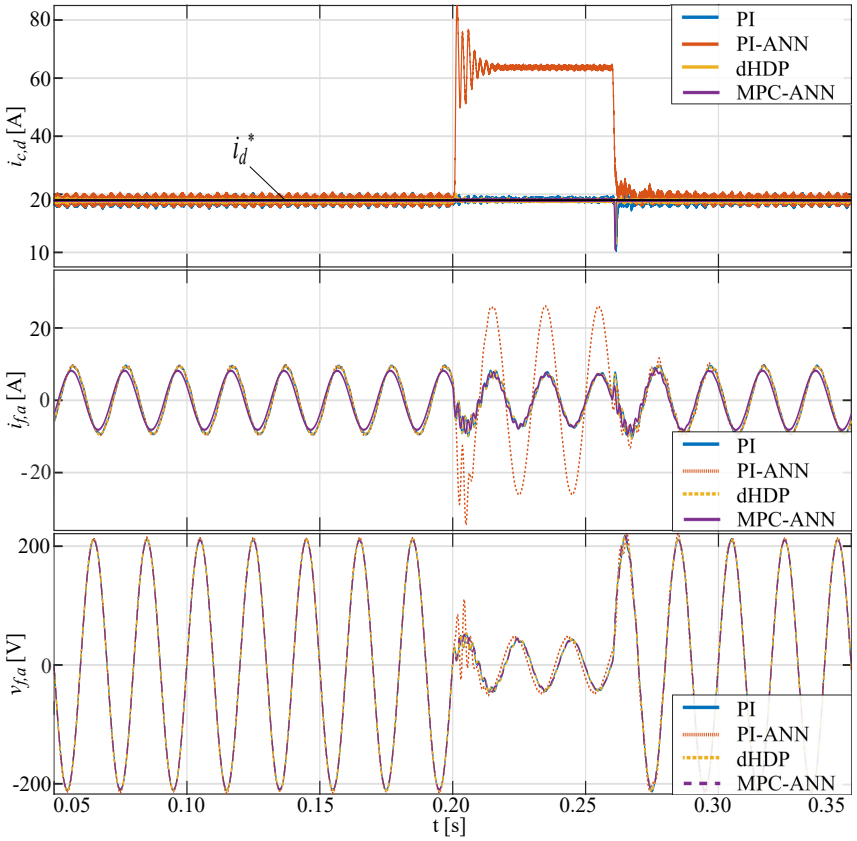


Figure 7.3: Simulation results of all controllers under grid-fault.

In Figure 7.3 it is seen that the PI-ANN controller is unable to tackle the fault, as the magnitude of $i_{c,d}$ is raised to a higher value and has large oscillations, even though the i_d^* magnitude is kept constant during the entire test case. Contrarily, the PI controller, the expert, operates at the expected level, where the deviation is seen to decrease after the fault is introduced. The same is observed for the dHDP and the MPC-ANN control schemes, which exhibit an even more accurate tracking. At the instant of short circuit clearing the $i_{c,d}$ magnitude has a momentarily dip when employing the PI, MPC-ANN and dHDP controllers. The MPC-ANN controller is fastest to restore the desired magnitude and the PI-controller is slowest. For the PI-ANN controller the $i_{c,d}$ magnitude starts to decrease after the fault is cleared. This decrease continues until the actual current magnitude restores the reference value. The current $i_{f,a}$ is also presented in Figure 7.3. The current waveform has a similar form for all controllers, however the waveform amplitude is

slightly lower for the MPC-ANN controller. During the short circuit fault the waveform amplitude is significantly increased for the PI-ANN controller. The voltage $v_{f,a}$ waveform is smooth during the short circuit, and appears identical for all controllers, except the PI-ANN controller which has momentarily large distortions at the introduction of the short circuit fault.

7.4 Parameter Uncertainty

The robustness of all the current-controllers is further verified under the parameter uncertainty test case. This test case considers a 20% increase and decrease of the filter inductance L_f and a 300% increase and decrease of the grid inductance. The i_d^* value is also changed to a different level at 0.2 s. The simulation results for the d-axis converter current is presented in Figure 7.4, phase-a of the filtered output current is presented in Figure 7.5 and phase-a of the filtered output voltage is presented in Figure 7.6.

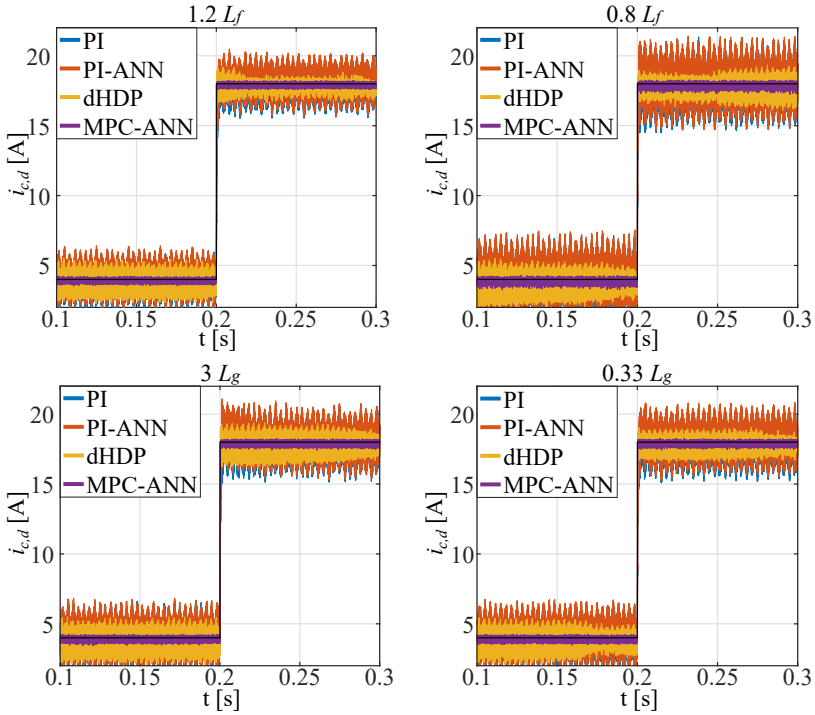


Figure 7.4: D-axis converter current under parameter uncertainty condition.

In Figure 7.4, the $i_{c,d}$ and i_d^* are presented for all the controllers for high and low L_f-values and L_g-values. The PI and PI-ANN controllers are most influenced by these parameter modifications, where the current $i_{c,d}$ is seen to have significantly larger oscillations around its reference component. The dHDP scheme comparatively performs better for all scenarios, however the tracking is seen to be affected by the parameter modifications. The performance of the MPC-ANN, on the other hand, appears unaffected by all parameter changes and clearly has the best performance.

In Figure 7.5 the current $i_{f,a}$ is presented for all controllers. Here, the dHDP and the MPC-ANN controllers exhibit the best performance. This is observed by the smooth current waveform, which appears unaffected by the different parameter modifications. The PI and PI-ANN controllers, on the other hand, produces a $i_{f,a}$ waveform which is distorted both before and after the reference current step change. The performance of the PI-ANN is largely comparable to the PI-controller, however for the high L_g-value the PI-ANN current waveform appear slightly more distorted.

In Figure 7.6 the voltage $v_{f,a}$ is presented for all controllers. Here, only the waveform of the MPC-ANN is visible as this was plotted last. It is worth mentioning that before presenting this result it was tested to change the line-style of the plotted voltage waveforms when using the different controllers. However, the voltage is so similar that its result became more confusing.

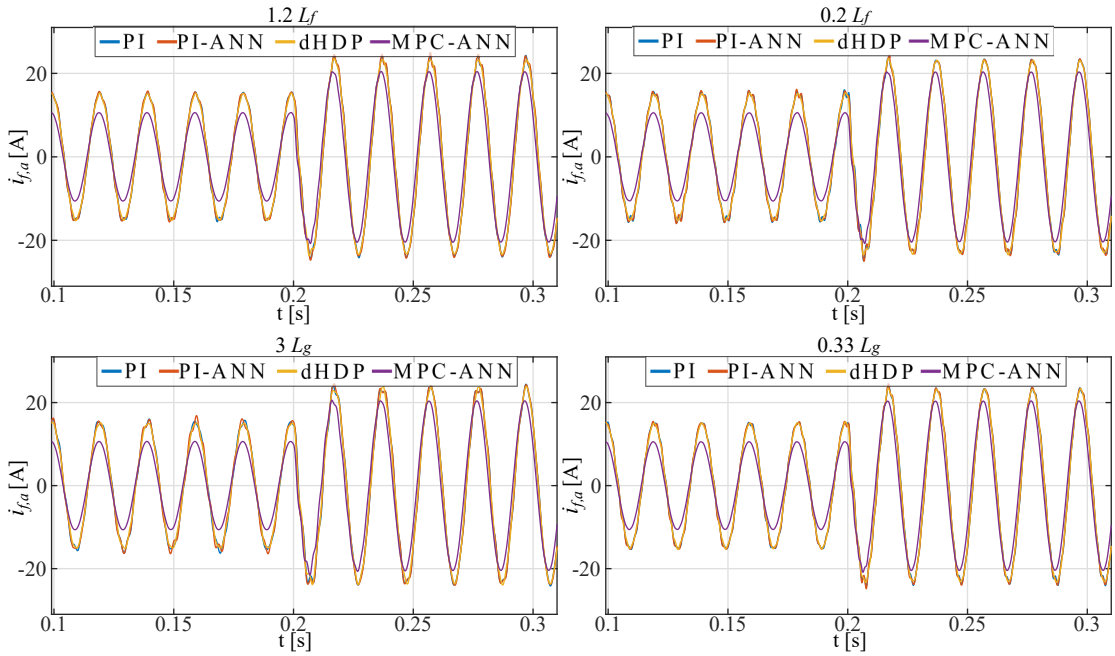


Figure 7.5: Phase a of filtered output current under parameter uncertainty condition.

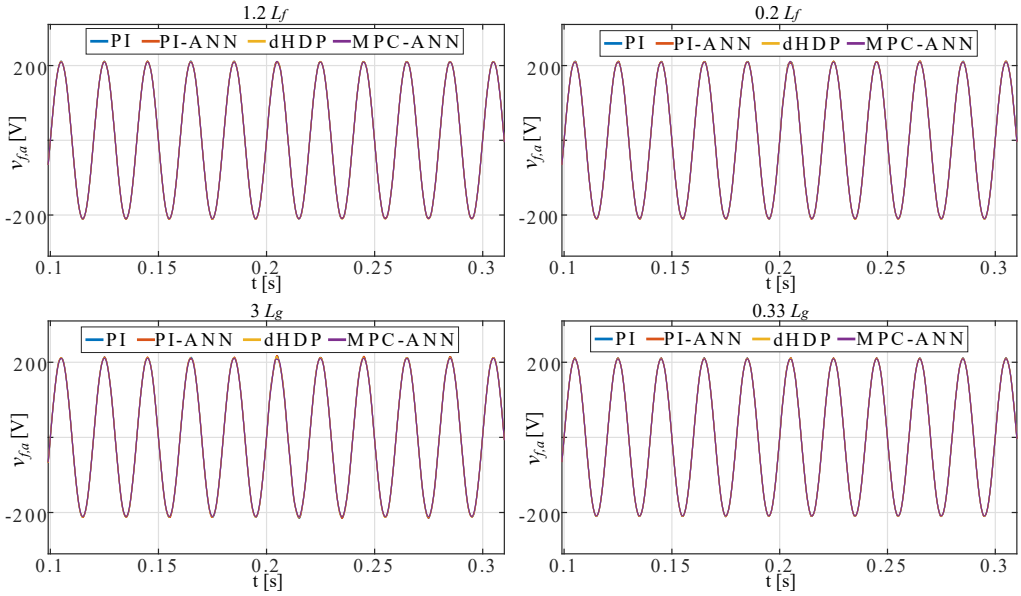


Figure 7.6: Phase a of filtered output voltage under parameter uncertainty condition.

7.5 Discussion

The simulation results showed that the PI-ANN controller shows no improvement compared to the conventional PI controller. For large disturbances, such as the grid fault, the PI-ANN even had a worse performance. The dHDP significantly improves the system performance compared to both the PI and PI-ANN controllers. The response speed is in all cases faster and the tracking is more accurate. However, the MPC-ANN remains the superior controller for all the scenarios.

Apart from this, there are two important remarks that must be highlighted here. Firstly, for all control methods, there are numerous user-defined parameters. This corresponds to error-prone control methods, which will, in almost any case, have the potential for improvement in terms of the hidden layer size, learning rates, reinforcement signal coefficients, etc. Secondly, the dHDP controller consists of heavy computations, mainly due to the inclusion of two while-loops, which must

be executed in each time step. The simulations took significantly more time than the other control methods when the dHDP scheme was executed in the Matlab/Simulink simulation environment. Unless the code is improved, this might pose an issue for the real-time hardware implementation.

8

Conclusion and Further Work

8.1 Conclusion

The connection of electronic power converters to the power grid is rapidly increasing due to a massive integration of renewable energy sources. As a result, the control methods for such grid-connected converters must ensure robustness, reliability, and fast dynamics. Conventional control methods, which depend on assumptions such as linearity and time invariance, are now being replaced by intelligent control methods. A literature review shows that among all AI technologies, the use of ANNs is a popular and powerful control approach.

This thesis starts with an investigation of AI technologies relevant for the grid-connected VSC, with the aim to improve the inner-loop current control. The investigation includes an overview and description of the AI technologies; fuzzy logic, supervised learning and reinforcement learning. Further, four ANN-based control methods are proposed and thoroughly described.

The first control method, the PI-ANN controller, consists of an FFNN which uses the conventional decoupled PI controller as an expert. That is, the inputs and outputs of the PI controller are stored during simulations and are fed to the network during the offline training phase. The network is fully trained offline using the LM algorithm. Both the PI controller and the PI-ANN controller are simulated using Matlab/Simulink. The performance analysis consider three different test cases; step-variation in reference current, short circuit at the distribution grid and parameter uncertainty by modifications in filter and grid inductance. The PI controller and the PI-ANN controller have comparable performances for the varying reference current and the parameter uncertainty test cases. For the short circuit test case the PI-ANN performs worse than the PI controller, and does not manage to track the reference current during the short circuit fault. The performance was validated through a comparison to the OPAL RT HIL simulation results.

The second control method studied is the MPC-ANN controller. Similar to the previous method, the controller consists of an FFNN which is trained using the LM algorithm, but now the inputs and outputs are stored using the simulated performance of the CCS-MPC as the expert. The performance of the MPC-ANN and the CCS-MPC controllers is evaluated using simulation results from the Matlab/Simulink environment. The results show that the MPC-ANN controller accurately tracks the reference current, and produces a filtered output current and a filtered output voltage with smooth waveforms and low distortions, for all aforementioned test cases. However, although being smooth, the filtered output current amplitude settles at a slightly lower magnitude for the MPC-ANN controller than what is seen for the CCS-MPC. The performance of both controllers is validated by comparison with OPAL RT simulation results.

The third controller is the DP-trained RNN-based controller. The DP algorithm consists of a combination of the LM algorithm and the FATT algorithm and is executed offline, before implementing the RNN as a controller. Because of the problem commonly known as exploding gradients, offline

training could not be completed and the proposed control method remains to be investigated.

The fourth and final controller is the dHDP-based controller. This consists of two FFNNs, which are pre-trained offline, and they continue learning after implementation. The controller is implemented in parallel to the decoupled PI controller. Simulation results obtained from Matlab/Simulink environment were used to evaluate the performance of the dHDP controller. The observed performance of the proposed controller is significantly improved compared to the conventional PI controller. The tracking is more accurate, the response speed is faster and the filtered output current and voltage have smoother waveforms.

A comparison of the performances of the conventional controller, the decoupled PI controller, and the ANN-based controllers, the PI-ANN, MPC-ANN and dHDP controllers, is also made based on the Matlab/Simulink simulation results. As for the above evaluations, this comparative analysis considers the same three test cases. Whilst the PI-ANN controller shows no improvement and has in some cases a slightly worse performance than the conventional PI controller, the dHDP controller has a significantly improved performance: the response is faster and the tracking is enhanced. The dHDP performance is also improved concerning changes in grid and filter inductance where the deviation between actual and reference converter current is reduced. However, some stability issues are observed due to the computational complexity this control method required. The MPC-ANN, on the other hand, exhibits a significantly improved performance in all cases and overall superior control.

8.2 Further Work

Considerable effort has been made to present the control methods and results in a comprehensible manner for students within the Department of Electric Power Engineering. The aim is to support as much as possible a continuation of this thesis' work and further activities could include the following topics:

1. Direct HDP Control

   The performance of the dHDP controller should be verified using HIL simulations. It is likely that the controller design must be further improved.

2. DP-trained RNN Control

   The RNN-based controller remains untested, and it would be interesting to compare its performance with the FFNN-based controllers. However, some modifications are necessary as the training is currently limited by the exploding gradients problem.

3. Other Intelligent Control methods

   Other control methods such as Q-learning remain relatively unexplored in converter control applications, and will therefore be interesting to investigate. Besides this, there also exist many suggestions for intelligent control methods which can provide the basis for developing new knowledge and perhaps achieving better performance than those studied in this thesis.

Appendix A

Performance Evaluation of ANN-Based Control for Grid-Connected Converter

In conjunction with this thesis a scientific paper was submitted for publication in the journal IEEE Access. The full paper is given in the proceeding pages.

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Performance Evaluation of ANN-Based Control of a Grid-Connected Converter with Different Training Datasets

ELLA GARDNER ØXNEVAD[1], PRABHAT RANJAN BANA[2], (STUDENT MEMBER, IEEE) AND MOHAMMAD AMIN,.[3], (Senior Member, IEEE)

[1,2,3]Department of Electric Power Engineering, Norwegian University of Science and Technology, Trondheim, Norway (e-mail: ellago@stud.ntnu.no)

Corresponding author: Mohammad Amin (e-mail: mohammad.amin@ntnu.no).

This paragraph of the first footnote will contain support information, including sponsor and financial support acknowledgment. For example, "This work was supported in part by the U.S. Department of Commerce under Grant BS123456."

ABSTRACT The conventional decoupled proportional-integral (PI) control of the voltage source converter (VSC) imposes several limitations on the controller performance due to assumptions such as linearity and time invariance. Among numerous control approaches, the use of an artificial neural network (ANN) draws attention in the research of artificial intelligence (AI) based converter control. In this paper, three different ANN-based control methods are considered for the inner current control of the VSC. The first controller is trained based on the simulation data from the conventional PI controller. The second controller is trained using the data from a model predictive control (MPC). The last controller consists of two networks, where both are trained using the direct heuristic dynamic programming (dHDP) algorithm. All controllers are implemented in Matlab/Simulink environment and are compared under different dynamic test cases. The ANN controller trained with MPC data (MPC-ANN) and the dHDP scheme enhance the system performance in terms of less oscillations, improved reference signal tracking, and robustness against parameter changes.

INDEX TERMS artificial neural network, model predictive controller, oscillations, supervisory learning, voltage source converter

I. INTRODUCTION

THE voltage source converter (VSC) is highly efficient, reliable, with a low-cost and high power density, and is becoming an integral part of the power grid. For this reason, the control of the VSC must ensure robustness, reliability, and fast dynamics. The use of proportional-integral (PI) control with pulse-width modulation (PWM) or space vector modulation (SVM) remains a popular control method due to its simplicity, strong adaptability, and reliability [1], [2]. This control method is designed in the synchronously rotating (dq) reference frame, where one PI controller is independently applied to the $d-$ and q-axis currents by use of appropriate decoupling terms [1], [3]. This coordinate transformation enables the successful application of linear PI controllers to dc components and results in a linear-time-invariant (LTI) system. Although this highly simplifies the control techniques, assumptions such as linearity and time-invariance limit the overall system performance. In response, a considerable effort has been made in recent years by the scientific and industrial communities to address the research challenge of improving VSC control [4].

The control methods can broadly be characterized as linear and non-linear. The non-linear methods exploit the fact that the averaged VSC model is non-linear by directly compensating for the non-linear dynamics. Non-linear control methods were first proposed in [5], and have been further developed in [6] and [7]. More robust non-linear control has been achieved using sliding-mode control (SMC) [2], [8]. Although such non-linear control strategies have been proven to improve the dynamic performance, unless advancements are implemented, they have the major disadvantage of producing a variable switching frequency, which creates resonance problems and reduces the overall efficiency [4], [9].

In the recent past, model predictive control (MPC) has evolved as a promising control method due to advantages

such as improved dynamic performance, easier handling of multiple-input, multiple-output (MIMO) cases, and increased flexibility compared to conventional methods [10], [11]. The MPC principle predicts the process output at future time instants and computes an optimal control signal that minimizes a cost function while ensuring that system constraints are met. The MPC strategies can be classified as continuous control set MPC (CCS-MPC) and finite control set MPC (FCS-MPC). The operating principles of these two types are principally similar but differ in the design of the cost function and the optimization stages [12]. In FCS-MPC, the discrete nature of the converter is exploited, and the control signals are switching signals which are directly applied to the converter without the use of an external modulator. On the other hand, the cost function and the control signals are continuous in the CCS-MPC scheme. The control signals are the modulation signals which are again processed through a PWM-modulator to generate the final switching pulses. Both methods have advantages and limitations, and the choice of method mainly depends on the application objectives [12]–[14].

Since the 1990s, artificial intelligence (AI) based control methods have gained substantial attention, and the number of applications is continuously increasing and improving [15]. Recently, numerous AI-based VSC control techniques have been proposed, which provide promising solutions to different objectives. Accounting to the learning approach, the AI concept can be classified as supervised learning (SL), unsupervised learning (UL) or reinforcement learning (RL). In SL, the goal is to learn the correct input-output mapping based on labeled data, and this learning method is mainly used for function fitting problems. UL aims to discover a pattern in unlabeled data sets, which is primarily used for data clustering. RL, from a control aspect, is broadly related to SL, except that it is not fed labeled data; instead, it learns by a reinforcement signal that either rewards or penalizes the actions taken when interacting with the environment. The controller will eventually learn the correct input-output mappings, aiming to maximize the rewards.

Moreover, the SL and RL approach is particularly interesting for developing control methods. For both learning approaches, the use of an artificial neural network (ANN) for control related problems is increasingly drawing attention. In SL, the ANN-based control methods mainly consist of training the network to approximate either an existing controller or an optimal one. In [4], [16] and [14] the former strategy has been done using the MPC as an expert. In [17] the latter method is done by use of a modified version of the Levenberg-Marquardt method. HDP methods commonly use a modified Bellman equation to train a network to map inputs to actions. Recently, the direct HDP (dHDP) method has shown promising control action and has been applied to a large power system stability control problem in [18] and for nonlinear tracking in [19], [20].

Furthermore, the ANN-based VSC control methods can be broadly classified into three categories [21]. Firstly, the ANN can be utilized to optimize the PWM stage. Secondly, the ANN be used as a replacement for the original PI controller. Additionally, the ANN can be employed as a supplementary part of the PI controller to achieve further improvement. It is worth highlighting that the last two categories are tested using both SL and RL in this work. The first controller replaces the conventional PI controller with an ANN, which is trained using the outputs of the original PI controller as targets. The next controller is based on a similar approach, but the MPC scheme is used as an expert instead of the PI controller. This control method is based on [16]. The last control method is based on the work done in [22] and consists of two networks, an actor and a critic, trained using the dHDP algorithm. The networks are implemented in parallel to the decoupled PI controllers.

Specifically, the control methods implemented are concerned with inner-loop current control of the VSC. Their performance is compared under different dynamic test scenarios. The inner current controller should track the reference current obtained from an outer control loop irrespective of the magnitude of disturbances. In this context, the tracking performance is observed in three different scenarios. First, without disturbances, only with a varying reference current. The second scenario considers the controller performance during and after a fault at the distribution grid. The third scenario is parameter uncertainty to test the robustness of the controller.

In line with the above discussion, the paper is organized as follows. First, Section II introduces the system model, including a schematic of the system and mathematical equation. Section III presents the control methods and gives the necessary details for implementation. Section IV presents the simulation results obtained from Matlab/Simulink and compares the performance of the three controllers along with the conventional PI controller. In Section V, the obtained results are summarised and the observed control performance is discussed. Finally, concluding remarks are provided in Section VI.

II. SYSTEM MODEL

The system to be controlled is a grid-connected three-phase two-level VSC. As illustrated in the top part of Fig.1, the system has three main components. The VSC, consisting of three phase legs each containing two switches. Secondly, The filter with inductance L_f, resistance R_f and capacitance C_f. Third, the distribution grid represented by inductance L_g, resistance R_g, and a voltage source v_g. Also shown in the figure are the converter output voltages v_c and current i_c, the filtered voltages v_f, and the DC-side capacitor C_{DC} and voltage V_{DC}. By using KVL on this system the model in (1) is obtained.

$$\boldsymbol{v}_{c,abc} = R_f \boldsymbol{i}_{abc} + L_f \frac{d\boldsymbol{i}_{abc}}{dt} + \boldsymbol{v}_{f,abc}, \tag{1}$$

The different control methods are implemented in different reference frames, as shown in Fig.1. The MPC is implemented in the stationary ($\alpha\beta$) reference frame, while the PI

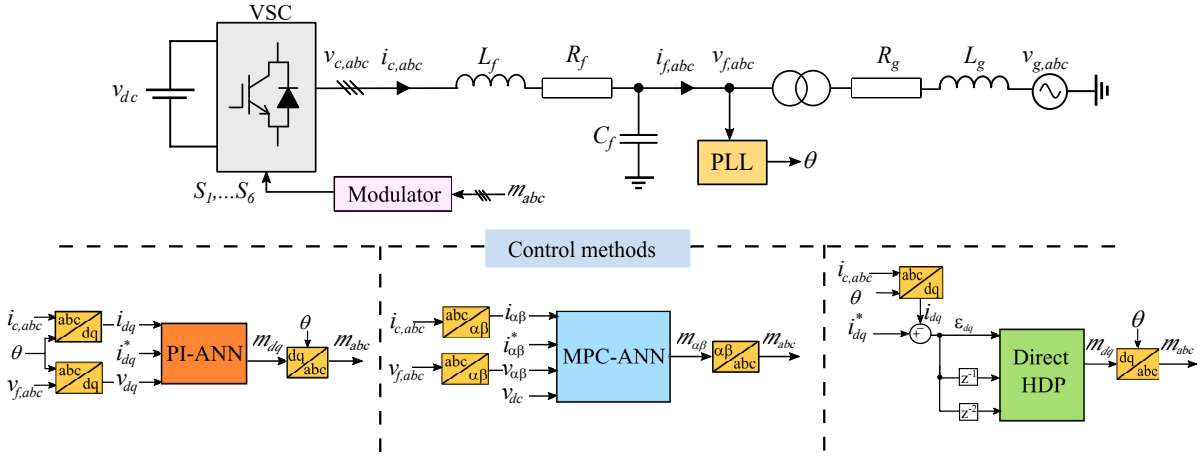


FIGURE 1. Implementation of the three control methods.

controller and the dHDP controller are implemented in the synchronous (dq) reference frame. The transformation from the three-phase components x_{abc} into the two components $x_{\alpha\beta}$ is done using the amplitude invariant Clarke transformation. The transformation from $\boldsymbol{x}_{abc}$ into $\boldsymbol{x}_{dq}$ is done using the Park transformation. The resulting system dynamics are given by (2) and (3) for the stationary and synchronous frame, respectively. Here ω is the angular frequency, which is obtained from the phase-locked loop (PLL).

$$\boldsymbol{v}_{c,\alpha\beta} = R_f \boldsymbol{i}_{\alpha\beta} + L_f \frac{d\boldsymbol{i}_{\alpha\beta}}{dt} + \boldsymbol{v}_{f,\alpha\beta}, \tag{2}$$

$$\begin{aligned} v_{c,d} &= R_f i_d - \omega L_f i_q + L_f \frac{di_d}{dt} + v_{f,d} \\ v_{c,q} &= R_f i_q + \omega L_f i_d + L_f \frac{di_q}{dt} + v_{f,q} \end{aligned} \tag{3}$$

The state space formulation is given by (4), where the states x, control inputs u and matrices $\mathbf{A}$ and $\mathbf{B}$ are given by (5) and (6) for $\alpha\beta$-reference frame and the dq-reference frame, respectively.

$$\dot{x} = \mathbf{A}\boldsymbol{x} + \mathbf{B}\left(\boldsymbol{v}_c - \boldsymbol{v}_f\right) = \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{u} \tag{4}$$

$$\mathbf{A}_{\alpha\beta} = \begin{bmatrix} -\frac{R_f}{L_f} & 0 \\ 0 & -\frac{R_f}{L_f} \end{bmatrix}, \mathbf{B}_{\alpha\beta} = \begin{bmatrix} \frac{1}{L_f} & 0 \\ 0 & \frac{1}{L_f} \end{bmatrix} \tag{5a}$$

$$\boldsymbol{x}_{\alpha\beta} = \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix}, \boldsymbol{u}_{\alpha\beta} = \begin{bmatrix} v_{c,\alpha} - v_{f,\alpha} \\ v_{c,\beta} - v_{f,\beta} \end{bmatrix} \tag{5b}$$

$$\mathbf{A}_{dq} = \begin{bmatrix} -\frac{R_f}{L_f} & \omega \\ -\omega & -\frac{R_f}{L_f} \end{bmatrix}, \mathbf{B}_{dq} = \begin{bmatrix} \frac{1}{L_f} & 0 \\ 0 & \frac{1}{L_f} \end{bmatrix} \tag{6a}$$

$$\boldsymbol{x}_{dq} = \begin{bmatrix} i_d \\ i_q \end{bmatrix}, \quad \boldsymbol{u}_{dq} = \begin{bmatrix} v_{c,d} - v_{f,d} \\ v_{c,q} - v_{f,q} \end{bmatrix} \tag{6b}$$

III. CONTROL DESIGN

In this section, the three control methods, PI-ANN, MPC-ANN, and dHDP, will be presented and described, including schematics illustrating the working principle of the controller, mathematical equations, and the neural network training methods.

Common for all controllers is that they all consist of an ANN, the purpose of which is to approximate either an optimal controller (dHDP) or an existing controller (PI-ANN and MPC-ANN). The ANNs are described considering the number of inputs, outputs, hidden layers, and the connection between them. In general, the ANNs can be separated into two categories, feedforward neural networks (FFNNs) and recurrent neural networks (RNNs). FFNNs only have connections forward to the next layer, resulting in a forward flow of information from input to output. In recurrent networks, there are also connections backwards, which form a cyclic flow of information. In this paper, only FFNNs are considered and will therefore be given more attention in the following sections.

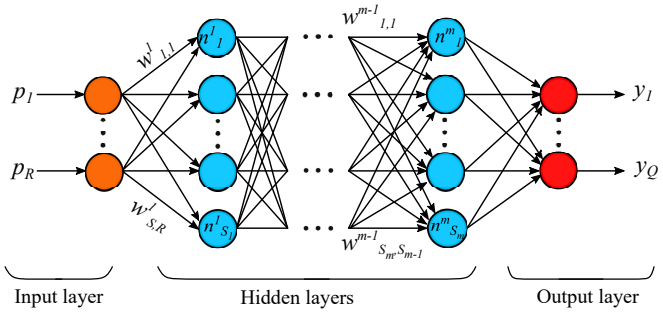


FIGURE 2. Fully-connected multi-layer feedforward neural network.

In Fig.2 a fully-connected multi-layer FFNN is depicted. It consists of R inputs, m hidden layers, where the first hidden layer contains S_1 neurons and the last hidden layer consists of S_m neurons, the total number of outputs is Q. If this network had 2 hidden layers, each containing 6 neurons,

we say the network structure is p-6-6-Q. In each layer the output is determined by the weight w, a bias b, and an activation function f, this is shown for the first neuron n_1^1 in (7). The connection from one layer to another is defined by an activation function. There are several different activation functions. In this work, the hyperbolic tangent function in (8) is chosen for the hidden layers, while a linear function is chosen for the output layer.

$$a_1^1 = f(w_{1,1}^1 p_1 + b_1) = f(n_1^1) \tag{7}$$

$$f(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}} \tag{8}$$

A. PI-ANN CONTROL MODEL

The structure of the PI-ANN control method is given in Fig. 3. First, the PI controller is used as an expert for generating the data needed for the training phase of the FFNN. The inputs and outputs of the PI controller are stored during simulations. Next, the network is trained using supervised learning, where the control inputs are used as network inputs, and the controller outputs are used as targets. After completing this offline-training, the network is implemented as a controller, replacing the original PI controller.

The rest of this section gives a brief description of the PI controller before describing the network structure and training algorithm.

1) Proportional Integral Control

The PI controller is implemented in the synchronous reference frame. The development of the control starts with a Laplace transformation of (3), as given in (9). In order to compensate for undesirable cross-coupling effects, the inductance terms $\omega L_f \mathbf{i}_{dq}$ and voltage terms v_{dq} are fed forwards, resulting in the mathematical controller description in (10).

$$\begin{aligned} i_d &= \frac{1}{sL_f + R_f}(v_{c,d} - v_d + \omega L_f i_q) \\ i_q &= \frac{1}{sL_f + R_f}(v_{c,q} - v_q - \omega L_f i_d) \end{aligned} \tag{9}$$

$$\begin{aligned} v_{c,d}^* &= \left(K_p + \frac{K_i}{s}\right)(i_d^* - i_d) + v_{f,d} - \omega L_f i_q \\ v_{c,q}^* &= \left(K_p + \frac{K_i}{s}\right)(i_q^* - i_q) + v_{f,q} + \omega L_f i_d \end{aligned} \tag{10}$$

The controller outputs $v_{c,dq}^*$ are then transformed back to three phase components $v_{c,abc}^*$ using an inverse Park transform. The resulting modulation indexes m_{abc} are obtained after normalizing $v_{c,abc}^*$ into the range $[-1, 1]$ as illustrated in Fig.3. The proportional and integral gains K_p and K_i are determined using modulus optimum, a technique based on cancelling the dominant time constant, while keeping the closed loop gain larger than unity for as high frequencies as possible. Modulus optimum is a popular tuning method for the inner current loop as it achieves fast response, for more information on this tuning procedure please refer to [23].

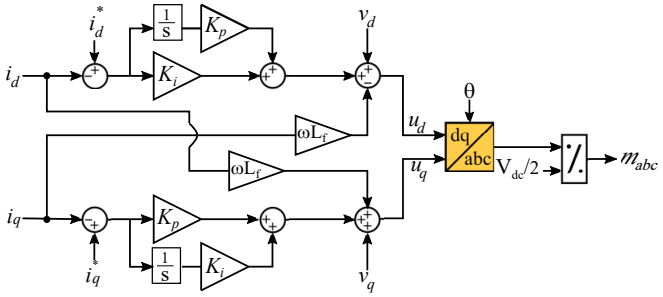


FIGURE 3. Decoupled PI control.

2) FFNN Structure

The number of inputs and outputs of the FFNN is the same as the number of inputs and outputs of the expert. Therefore, as is shown in Fig.3, the FFNN takes 6 inputs, i_{dq}, i_{dq}^* and v_{dq} and outputs the two control signals u_{dq}. Different sizes of hidden layers are tested considering the tracking performance of the implemented network controller during simulations. A range of hidden layer sizes starting from 6 to 30 neurons were tested, the best performance was obtained with the structure 6-10-2.

3) Training Algorithm

The weights and biases in an ANN are determined during training. The data obtained during simulations using the original PI controller is stored in a look-up table, where the sampled control inputs are used as inputs to the network and the outputs are used as targets for the network outputs, as is illustrated in Fig.4. The goal of the training is thus to update the network weights such that the outputs of the network are as close as possible to the targets. In this paper, both the PI-FFNN and the MPC-FFNN are trained using the Levenberg Marquandt (LM) algorithm.

The LM algorithm was developed to solve nonlinear least square problems and is based on a combination of the steepest descent (SD) method and the Gauss-Newton method. The LM algorithm switches between the aforementioned algorithms depending on whether the model is linear in its parameters or not. This way when the parameters are distant from the optimal values, the problem becomes nonlinear and the steepest descent method is employed. Conversely, when the parameters become close to their optimal values, the problem is close to quadratic, and the Gauss-Newton method is employed [24].

The LM algorithm measures the network performance during training as the sum of squared errors (SSE) between the network outputs y and targets t_n at sample n, as given in the cost function (11) for a total of N samples, R inputs, M layers and S_i neurons in each layer i. The last summation is the sum over all elements in the error vector v_i, which is

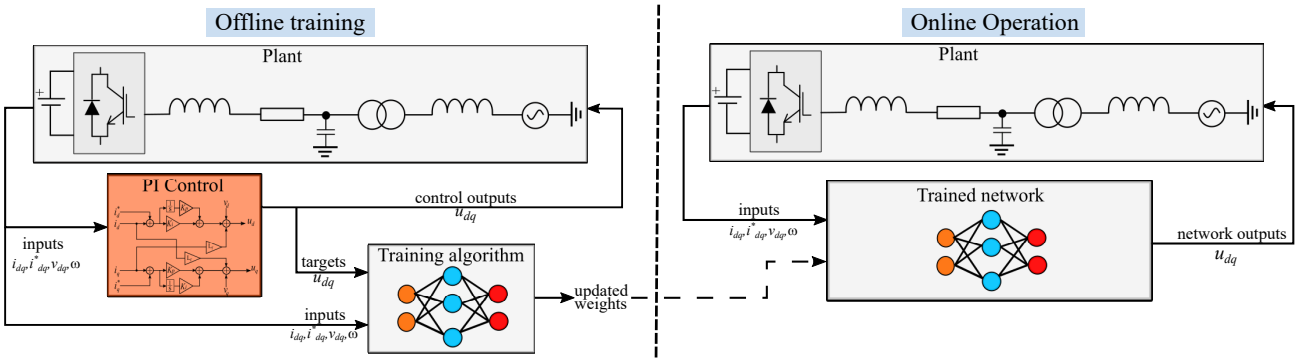


FIGURE 4. Offline and online phase of PI-ANN controller.

defined in (12).

$$\begin{aligned} C &= \sum_{n=1}^{N} (\boldsymbol{y} - \boldsymbol{t}_n)^\top (\boldsymbol{y} - \boldsymbol{t}_n) \\ &= \sum_{n=1}^{N} \boldsymbol{e}_n^\top \boldsymbol{e}_n = \sum_{i=1}^{N \times S_M} \boldsymbol{v}_i^2 \end{aligned} \tag{11}$$

$$\begin{aligned} \boldsymbol{v}^\top &= \begin{bmatrix} v_1 & v_2 & \dots & v_N & v_{N+1} & \dots & v_{N \times S_M} \end{bmatrix} \\ &= \begin{bmatrix} e_{1,1} & e_{2,1} & \dots & e_{S_M,1} & e_{1,2} & \dots & e_{S_M,N} \end{bmatrix} \end{aligned} \tag{12}$$

The update of weight and biases is thus based on minimizing (11). This is described in (13) and (14), where the new weight W_{new} is the sum of the current weight W_{old} and the change in weight δW for which the Jacobian $\mathbf{J}$ decreases. The Jacobian is defined in (15) as the derivative of the cost function with respect to the network parameters. The parameter vector x given by (16) contains all the weights and biases in the network. The switching between the SD method and the Gauss-Newton method is determined by the inverse of the learning rate mu. If $\mu >> \mathbf{J}^\top \mathbf{J}$, then (13) approximates to $-\frac{1}{\mu}\mathbf{J}^\top \boldsymbol{v}$ which is the steepest descent update. If μ, on the other hand, is small we get approximately $-(\mathbf{J}^\top \mathbf{J})^{-1}\mathbf{J}^\top \boldsymbol{v}$, a Gauss-Newton update. In LM the cost function C is evaluated after each iteration, if it is not decreased, then μ is increased corresponding to a lower learning rate and a smaller step. If the performance has improved μ is decreased, and a larger step is taken. In this paper μ is initially set to 0.001 and decreased and increased by 10. (16).

$$\mathbf{W}_{new} = \mathbf{W}_{old} + \Delta \mathbf{W} \tag{13}$$

$$\Delta \mathbf{W} = -(\mathbf{J}^\top \mathbf{J} + \mu \mathbf{I})^{-1} \mathbf{J}^\top \boldsymbol{v} \tag{14}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{15}$$

$$\begin{aligned} \mathbf{x}^\top &= \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \\ &= \begin{bmatrix} w_{1,1}^1 & \dots & b_{S_1}^1 & w_{1,1}^2 & \dots & b_{S_M}^M \end{bmatrix} \end{aligned} \tag{16}$$

The training continues until one of three stopping criteria are met,

1) The maximum number of epochs is reached.
2) The maximum value of μ is reached.
3) The gradient of the cost function $\frac{\partial C}{\partial x}$ reaches a minimum value.

B. MPC-ANN CONTROL MODEL

The methodology of the MPC-ANN controller is identical to the PI-ANN controller shown in Fig.4, except now the FFNN will be trained using the MPC as an expert. As illustrated in Fig.1 and previously mentioned in Section III, the MPC is implemented in the $\alpha\beta$-frame. The LM algorithm described in Section III-A3 is used for training the network. The FFNN will have the same number of inputs and outputs as the expert, which in this case correspond to 7 inputs ($i_{\alpha\beta}$,$i^*_{\alpha\beta}$,$v_{\alpha\beta}$, v_{dc}) and 3 outputs (m_{abc}). The same hidden layer sizes as stated above were tested for the MPC-ANN and the best performance was obtained with the structure 7-20-3. The training of the network was done using the LM algorithm.

In this paper, the CCS approach is employed. Here the optimal control signal is calculated based on a system model. The cost function is defined as a single-variable function based on the controller-goal. The optimal control signal is defined by minimizing this cost function while including the model predictions. Once this control signal is determined, it is sent to the modulator, which generates the switching commands. Since the MPC replaces the PI controller, the cost function is defined as the error between measured and reference current, as given by (17). Here $\mathbf{Q}$ is the weight matrix, choosing equal importance for both i_α and i_β corresponds to the weight in (18). Minimizing C consists of finding the control signal for which $\partial J/\partial \boldsymbol{v}_{c,\alpha\beta} = 0$. By replacing $i_{\alpha\beta}[k+1]$ with the model defined in (5), the optimal control for which C is minimized is defined by (19).

$$\min_{\boldsymbol{v}_{c,\alpha\beta}} C = (\boldsymbol{i}^*_{\alpha\beta} - \boldsymbol{i}_{\alpha\beta}[k+1])\mathbf{Q}(\boldsymbol{i}^*_{\alpha\beta} - \boldsymbol{i}_{\alpha\beta}[k+1]) \tag{17}$$

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{18}$$

$$\mathbf{v}_{c,\alpha\beta}[k] = \left(\mathbf{B}\mathbf{B}^\top\right)^{-1}\left(\mathbf{B}^\top \mathbf{i}^*_{\alpha\beta} - \mathbf{B}^\top \mathbf{A}\mathbf{i}_{\alpha\beta}[k+1] - \mathbf{B}^\top \mathbf{B}\mathbf{v}_{f,\alpha\beta}[k]\right) \quad (19)$$

The CCS-MPC can have an extended horizon of several time steps, in this article a prediction horizon of 4 time steps gave the best results and is therefore chosen. For an extended horizon the model must be augmented as given by (20).

$$\boldsymbol{x}_{aug} = \begin{bmatrix}\boldsymbol{x}[k+1] & \boldsymbol{x}[k+2] & \dots & x[k+N]\end{bmatrix}^\top, \quad (20a)$$

$$\boldsymbol{u}_{aug} = \begin{bmatrix}\boldsymbol{u}[k] & \boldsymbol{u}[k+1] & \dots & \boldsymbol{u}[k+N-1]\end{bmatrix}^\top \quad (20b)$$

$$\mathbf{A}_{aug} = \begin{bmatrix}\mathbf{A} & \mathbf{A}^2 & \dots & \mathbf{A}^N\end{bmatrix}^\top \quad (20c)$$

$$\mathbf{B}_{aug} = \begin{bmatrix} \mathbf{B} & \mathbf{0} & & \dots & \mathbf{0} \\ \mathbf{AB} & \mathbf{B} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{A}^2\mathbf{B} & \mathbf{AB} & \mathbf{B} & \dots & \mathbf{0} \\ \vdots & & \ddots & & \vdots \\ \mathbf{A}^{N-1}\mathbf{B} & \mathbf{A}^{N-2}\mathbf{B} & \dots & \mathbf{AB} & \mathbf{B} \end{bmatrix} \quad (20d)$$

C. DIRECT HDP CONTROL MODEL

Heuristic dynamic programming (HDP) is one of the most basic and widely used structures within the field of adaptive dynamic programming (ADP) [25], where ADP is the family of methods that seek to find an (approximately) optimal control policy for a stochastic process whose model depends on unknown parameters [26]. The dHDP estimates the controller parameters directly, unlike indirect versions which estimate model parameters before computing the control signals [27], [28]. The direct version avoids estimating the process, and enables the use of a model-free controller which is robust with respect to model uncertainties. This is illustrated in Fig.5 which illustrates the working of the dHDP method. As illustrated by the orange arrows ($x(t)$), the information from the process is directly used to find appropriate controller parameters.

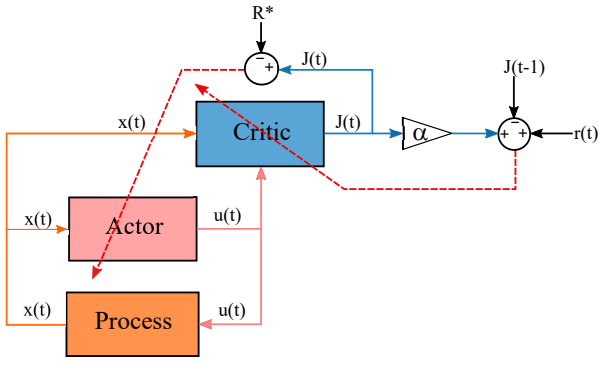


FIGURE 5. The direct HDP structure.

The dHDP controller is implemented in parallel to the decoupled PI control, as illustrated in Fig.6. The errors $\varepsilon_{dq} = i^*_{dq} - i_{dq}$ are inputs to both the dHDP and the PI controllers. The output of the dHDP is added as a supplement to the output of the original PI controllers. The resulting control signals u^*_{dq} are then normalized (divided by $\frac{V_{dc}}{2}$) and transformed back to three-phase components, as was shown for the decoupled PI control in Fig.3.

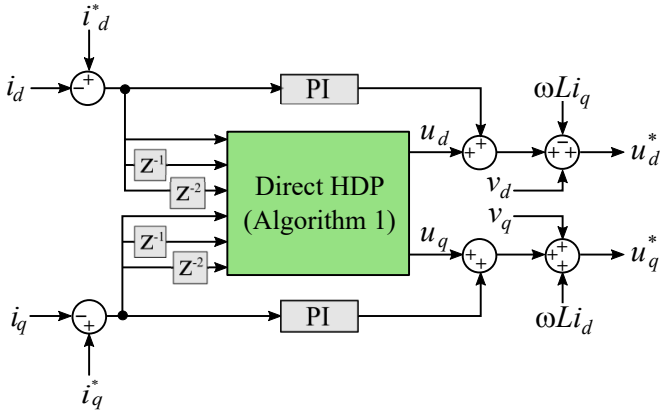


FIGURE 6. The implementation of the dHDP controller.

1) Cost-to-Go Function

The control design starts with defining J which describes the cost-to-go of inputs $\boldsymbol{x}(t)$ for a utility function U_c which measures the one-step cost or reward of control, using a discount rate $\gamma \in 0, 1$. The objective of any DP-based control is to select a control signal $\boldsymbol{u}(t)$ which minimizes this cost according Bellman's optimality principle. This leads to an optimal control signal as given in (22) to (23). These equations are approximated in an iterative manner using two FFNNs, namely the actor and the critic. As their names imply the actor approximates the actions u and the critic evaluates the actor's performance by approximating J.

$$J[x(t), t] = \sum_{k=t}^{\infty} \gamma^{k-t} U[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \quad (21)$$

$$J^*[x(t)] = \min_{\boldsymbol{u}(t)}\{J^*[\boldsymbol{x}(t+1)] + r[\boldsymbol{x}(t)] - U_c\} \quad (22)$$

$$\boldsymbol{u}^*(x(t)) = arg\min_{\boldsymbol{u}(t)} J^*[\boldsymbol{x}(t)] \quad (23)$$

2) Structure of Critic and Actor

As illustrated in Fig.5, the critic takes both the control signals generated by the actor and the error signals obtained from the system as inputs, which in total becomes 8 inputs (u_{dq}, $\varepsilon_{dq}(t)$, $\varepsilon_{dq}(t-1)$, $\varepsilon_{dq}(t-2)$). The input to the actor consists of only the error signals from the system and thus has 6 inputs. As proposed in [22], one hidden layer with 12 neurons is chosen for both networks. The output of the critic is the approximated cost-to-go function J, while the actor outputs the optimal control signals u^*_d and u^*_q, as described by (24) and (25). In summary, the structures of the critic and actor are 8-12-1 and 6-12-2, respectively.

$$J(t) = NN_c(x(t), u(t)) \quad (24)$$

$$u_{dq} = NN_a(x(t)) \quad (25)$$

The aim of the critic is to approximate the *value* function, also known as the cost-to-go function. This is done by minimizing the objective function E_c defined by (26). The prediction error e_c reflects the difference in the current

and previous cost-to-go functions $J(t)$ and $J(t-1)$, in addition to the amount of reinforcement that was currently implemented $r(t)$, as described in (27). The goal is that J converges towards the optimal cost-to-go J^* and keeping the reinforcement signal at a minimum, only then an appropriate value function is obtained. As given in (28) the reinforcement signal is determined by the amount of deviation between reference and measured currents, ε_{dq}, for the current and previous time steps. The deviation at previous time steps, $\varepsilon_{dq}(t-1)$ and $\varepsilon_{dq}(t-2)$, is included to limit the strength of the control signals during steady state, while allowing stronger control signals during transient conditions. The coefficients a_1, a_2 and a_3 are kept constant and are chosen equal to those proposed in [22].

$$E_c(t) = \frac{1}{2} e_c(t)^2 \tag{26}$$

$$e_c(t) = \alpha J(t) - J(t-1) + r(t) \tag{27}$$

$$r(t) = -(a_1 \varepsilon_{dq}^2(t) + a_2 \varepsilon_{dq}^2(t-1) + a_3 \varepsilon_{dq}^2(t-2)) \tag{28}$$

The actor approximates the *policy* by selecting appropriate actions based on the cost-to-go J, this is given by (29) and (30).

$$E_a(t) = \frac{1}{2} e_a^2(t) \tag{29}$$

$$e_a(t) = J(t) \tag{30}$$

3) Direct HDP Training Algorithm

Unlike the two previous control methods which employ techniques within supervised learning, the dHDP controller falls within reinforcement learning (RL), as its network learns by interacting with its environment. More specifically the training of the networks are done in two stages, first offline and second online. For both stages the weights are updated based on gradient descent, given by (31) and (32). The partial derivative $\frac{\partial E}{\partial w}$ represents the change in the objective function based on the change in the weights and is computed using (33) and (34) for the critic and actor, respectively. The learning rate l is a positive and small number, unlike the learning rate μ used in the LM algorithm, it remains constant during the whole training period.

$$w_{new} = w_{old} + \Delta w \tag{31}$$

$$\Delta w = -l \frac{\partial E}{\partial w} \tag{32}$$

$$\frac{\partial E_c}{\partial w_c} = \frac{\partial E_c}{\partial J} \frac{\partial J}{\partial w_c} \tag{33}$$

$$\frac{\partial E_a}{\partial w_a} = \frac{\partial E_a}{\partial J} \frac{\partial J}{\partial u_{dq}} \frac{\partial u_{dq}}{\partial w_a} \tag{34}$$

In offline training, the weights are initialized randomly for both the critic and actor. For the online training, the weights obtained offline are used as initial weights. Despite this, the training algorithm remains the same.

The first step of the algorithm is to initialize the weights for the critic W_c and actor W_a, the control signal u_{dq} and the cost-to-go J as is shown in line 1 of Algorithm 1. The second step is to calculate the control signal u_{dq} using the action network and the cost-to-go using the critic network. After implementing the control signals, the input at the next time step $\boldsymbol{x}(t+1)$ and the reinforcement signal $r(t)$ is obtained, as is seen in lines 3-5. Based on this data the training of the network is started and consists of going through (26) until (34). The training continues until a maximum number of iterations k^{max} is reached, or if the prediction error falls below a satisfactory limit E^{min}.

Algorithm 1 Direct HDP training algorithm

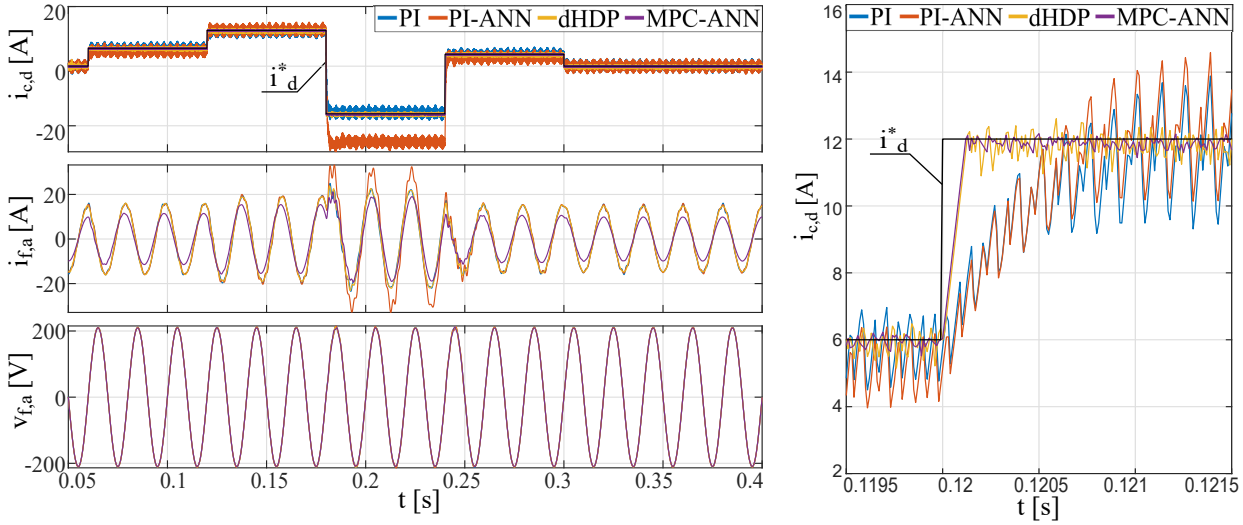
1: initialize W_c, W_a, $u_{dq} \leftarrow \begin{bmatrix} 0 & 0 \end{bmatrix}^\top$, $J \leftarrow 0$, $k_a \leftarrow 1$, $k_c \leftarrow 1$
2: **repeat**
3: $\quad \boldsymbol{u}_{dq}(t) \leftarrow NN_a(\boldsymbol{x}(t))$
4: $\quad J(t) \leftarrow NN_c(\boldsymbol{x}(t))$
5: $\quad$ insert $\boldsymbol{u}_{dq}$, get $\boldsymbol{x}(t+1)$, compute $r(t)$
6: $\quad$ **while** $E_c \geq E_c^{min}$ and $k_c \leq k_c^{max}$ **do**
7: $\quad\quad e_c(t) \leftarrow \alpha J(t) - J(t-1) + r(t)$
8: $\quad\quad E_c(t) \leftarrow \frac{1}{2} e_c^2(t)$
9: $\quad\quad \Delta W_c \leftarrow -l \frac{\partial E_c}{\partial W_c}$
10: $\quad\quad W_c \leftarrow W_c + \Delta W_c$
11: $\quad\quad k_c \leftarrow k_c + 1$
12: $\quad$ **end while**
13: $\quad$ **while** $E_a \geq E_a^{min}$ and $k_a \leq k_a^{max}$ **do**
14: $\quad\quad e_a(t) \leftarrow J(t)$
15: $\quad\quad E_a(t) \leftarrow \frac{1}{2} e_a^2(t)$
16: $\quad\quad \Delta W_a \leftarrow -l \frac{\partial E_a}{\partial W_a}$
17: $\quad\quad W_a \leftarrow W_a + \Delta W_a$
18: $\quad\quad k_a \leftarrow k_a + 1$
19: $\quad$ **end while**
20: $\quad \boldsymbol{x}(t) \leftarrow \boldsymbol{x}(t+1)$
21: $\quad J(t-1) \leftarrow J(t)$
22: **until** simulation is finished

IV. PERFORMANCE EVALUATION

The grid-connected VSC system, as shown in Fig. 1 is implemented in the Matlab/Simulink environment. All the controllers, as mentioned earlier, are implemented for a switching model of VSC considering the system and control parameters given in Table 1 and 2, respectively.

A. CURRENT REFERENCE TRACKING

A simulation is carried out for all the controllers by varying the d-axis reference current magnitude (i_d^*) continuously and step-wise. The obtained results under this test case are presented in Fig. 7. The successful implementation of all the controllers can be verified from Fig. 7(a), as the actual d-axis current ($i_{c,d}$) is accurately tracking the corresponding reference component. This is better illustrated in the zoomed version of this figure, as shown in Fig. 7(b). Further, phase-a of the output current $i_{f,a}$ and output voltage $v_{f,a}$ for all the controllers are also included in Fig. 7(a). It can be observed



(a) Variation of current reference

(b) Close-up of $i_{c,d}$

FIGURE 7. Performance evaluation of the controllers under reference current variation with corresponding zoomed view

TABLE 1. Parameters for the grid-connected VSC system

Symbol	Description	Value
V_g	Grid voltage	280 V
V_{dc}	DC-link voltage	500 V
P	Rated power	10 kW
f_{sw}	Switching frequency	8000 Hz
L_f	Filter inductance	1.55 mH
R_f	Filter resistance	10 mΩ
C_f	Filter capacitance	75μF
L_g	Grid inductance	0.266 mH
R_g	Grid resistance	11.9 mΩ
C_{dc}	DC-link capacitance	3 mF
T_s	Sampling time	10 μs

TABLE 2. Controller parameters

Symbol	Description	Value
PI		
K_p	Proportional Gain	4.13
K_i	Integral Gain	25.83
PI-ANN and MPC-ANN		
$Epoch^{max}$	Maximum number of epochs	1000
$\alpha = \frac{1}{\mu}$	Learning rate	0.01
α^{min}	Minimum learning rate	10^{-10}
α_{inc}	Increase of learning rate	10
α_{dec}	Decrease of learning rate	0.1
Direct HDP (online,offline)		
l	Learning Rate (online, offline)	0.001, 0.01
k_a	Iteration, actor (online, offline)	40, 10
k_c	Iteration, critic (online, offline)	50, 20
E_a^{min}	Minimum error, actor (online, offline)	10^{-6}
E_c^{min}	Minimum error, critic (online, offline)	10^{-7}
a_1, a_2, a_3	Reinforcement coefficients	0.4, 0.2, 0.04
α	Discount factor	0.1

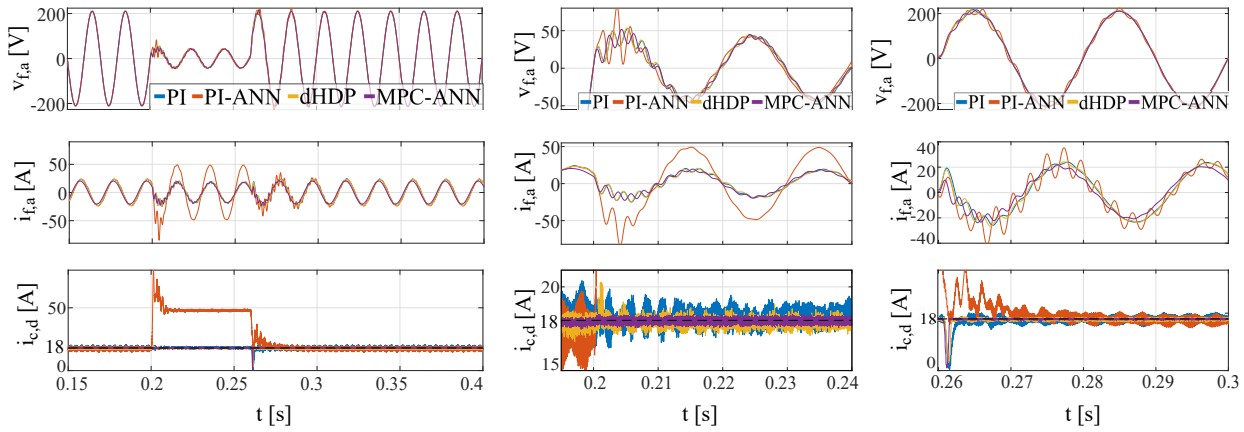
that the MPC-ANN and dHDP schemes are comparatively performing superior to the PI and PI-ANN controllers in maintaining a stable and sinusoidal $i_{f,a}$. Moreover, the $i_{f,a}$ obtained with PI-ANN is momentarily distorted when it faces a larger deviation in (i_d^*) magnitude. Besides, the $v_{f,a}$ waveform is satisfactorily maintained at the desired level by all the controllers.

B. ANALYSIS DURING GRID-SIDE FAULT

The performance of all the controllers is also tested under grid-side fault conditions by reducing the grid voltage amplitude symmetrically to 20%. As presented in Fig. 8, the fault is introduced at 0.2 s and lasts up to 3 cycles. In this context, the inability of the PI-ANN controller to tackle the fault can be seen from Fig. 8(a), as the magnitude of $i_{c,d}$ is raised to a higher value even though i_d^* magnitude is kept constant at 18A during the entire test case. Contrarily, the PI and dHDP control scheme operate at the expected level with lesser distortions, which can also be verified from the zoomed view of Fig. 8(a) presented in Fig. 8(b-c). Nonetheless, the ANN-MPC scheme performs superior to the other controllers for this case study.

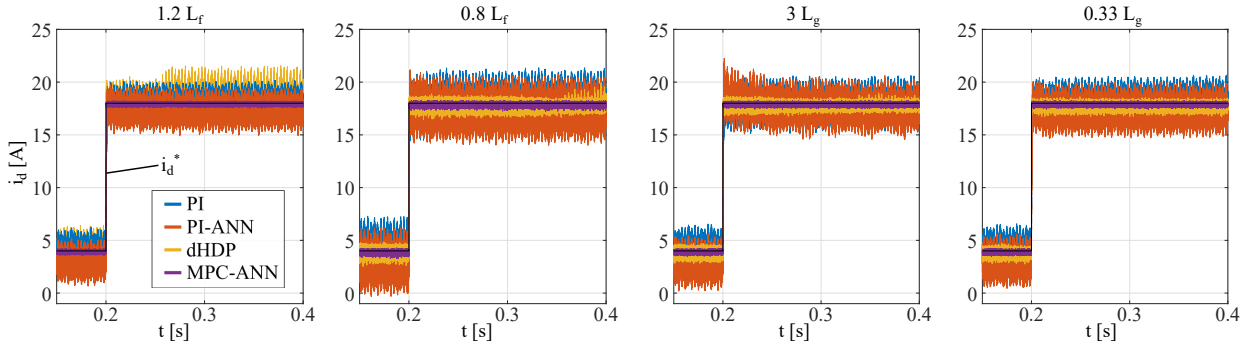
C. PARAMETER UNCERTAINTY

The robustness of all the current-controllers is further verified under the parameter uncertainty test case. In Fig.9, the $i_{c,d}$ and i_d^* are presented for all the controllers for an increased and decreased filter inductance L_f and grid inductance L_g. The i_d^* value is also changed to a different level at 0.2 s. It can be visualised that the performance of the PI and PI-ANN controller is largely influenced with this abnormality, $i_{c,d}$ waveform oscillates around its reference component. The dHDP scheme comparatively performs better for most of the scenarios, except when the L_f is increased. Note that the most oscillatory response is plotted first, which is the dHDP in this case. The performance of the MPC-ANN, on the other hand, appears unaffected by all parameter changes and clearly has the best performance.

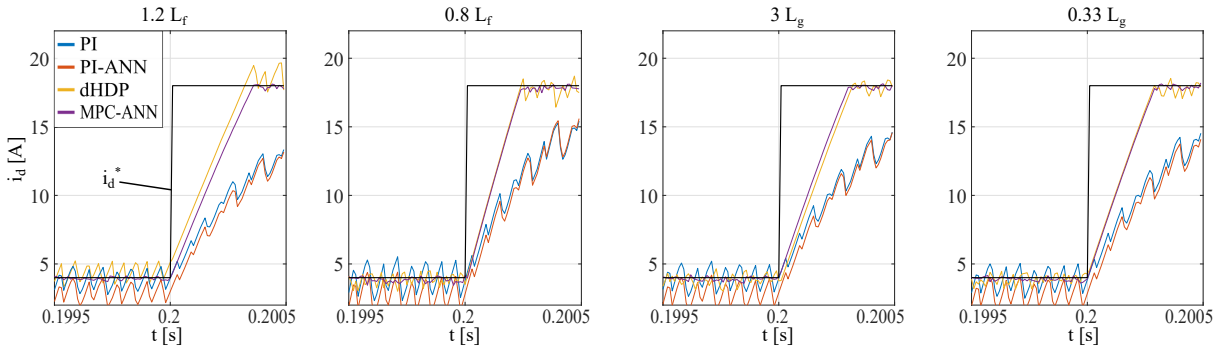


(a) Short circuit at 0.2s (b) Voltage step-down (c) Voltage step.up

FIGURE 8. Simulation results of the controllers under grid-fault: (b) and (c) are corresponding zoomed view.



(a) Tracking performance for modified L_f and L_g



(b) Close-up

FIGURE 9. Current tracking performance of all the controllers under parameter uncertainty condition with corresponding zoomed view

V. DISCUSSION

The above simulation study is summarized in Table 3, which includes the total harmonic distortion (THD) of the output current waveform for each controller under every considered scenario after steady-state is established. The dHDP significantly improves the system performance compared to the PI and PI-ANN controller. However, the MPC-ANN remains the superior controller for all the scenarios. Apart from this, there are two important remarks that must be highlighted here. Firstly, for all control methods, there are numerous user-defined parameters. This corresponds to error-prone control methods, which will, in almost any case, have the potential for improvement in terms of the hidden layer size, learning rates, reinforcement signal coefficients, etc. This remark is specifically important for the dHDP, which shows an improved performance by changing the discount factor value based on the trial and error method. Secondly, the dHDP controller consists of heavy computations, mainly due to the inclusion of two while-loops, which must be executed in each time step. The simulations took significantly more time than the other control methods when dHDP scheme was executed in the Simulation environment. Unless the code is

TABLE 3. Comparative analysis in terms of current THD [%]

Scenario	PI	PI-ANN	MPC-ANN	dHDP
Varying reference current	4.16	3.98	0.4	1.19
Short circuit	2.83	3.56	0.17	1.16
Increased grid inductance	4.52	32.77	0.17	2.15
Decreased grid inductance	2.90	2.91	0.18	1.31
Increased filter inductance	3.39	4.95	0.15	1.00
Decreased filter inductance	3.93	4.69	0.25	2.28

improved, this might pose an issue for the real-time hardware implementation.

VI. CONCLUSION

In this paper, three different ANN-based controllers are designed as a current controller for the grid-connected VSC systems. Two of the control methods use an expert,$i.e.$, the PI controller and the MPC, to train the networks. The other controller is based on dHDP and consists of two networks which are first trained offline but continue learning after being implemented. The controllers are implemented in Simulink and the performance is extensively compared. The comparative analysis consists of three different scenarios; step-variation in reference current, fault at distribution grid and change in filter and grid inductance. The dHDP has improved performance compared to the PI and PI-ANN controller in terms of faster response and improved tracking. Its performance is also significantly improved for parameter uncertainty conditions, considering the fact that the discount factor is manually tuned. However, the DHDP scheme is computationally heavy and generates even higher oscillations than the PI and PI-ANN controller for some specific test scenarios. The MPC-ANN, on the other hand, significantly improves the system performance in every aspect.

References

[1] R. D. Pereira, M. Veronesi, A. Visioli, J. E. Normey-Rico, and B. C. Torrico, "Implementation and test of a new autotuning method for pid controllers of tito processes," *Control Engineering Practice*, vol. 58, pp. 171–185, 2017.

[2] J. Liu, Y. Yin, W. Luo, S. Vazquez, L. G. Franquelo, and L. Wu, "Sliding mode control of a three-phase ac/dc voltage source converter under unknown load conditions: Industry applications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 10, pp. 1771–1780, 2017.

[3] Y. Gui, X. Wang, F. Blaabjerg, and D. Pan, "Control of grid-connected voltage-source converters: The relationship between direct-power control and vector-current control," *IEEE Industrial Electronics Magazine*, vol. 13, no. 2, pp. 31–40, 2019.

[4] I. S. Mohamed, S. Rovetta, T. D. Do, T. Dragicević, and A. A. Z. Diab, "A neural-network-based model predictive control of three-phase inverter with an output LC filter," *IEEE Access*, vol. 7, pp. 124 737–124 749, 2019.

[5] P. Rioual, H. Pouliquen, and J. Louis, "Non linear control of pwm rectifier by state feedback linearization and exact pwm control," in *Proceedings of 1994 Power Electronics Specialist Conference-PESC'94*, IEEE, vol. 2, 1994, pp. 1095–1102.

[6] E. Song, A. F. Lynch, and V. Dinavahi, "Experimental validation of nonlinear control for a voltage source converter," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1135–1144, 2009.

[7] R. M. Milasi, "Adaptive and nonlinear control of a voltage source converter," Ph.D. dissertation, University of Alberta, 2012.

[8] K. D. Young, V. I. Utkin, and U. Ozguner, "A control engineer's guide to sliding mode control," *IEEE transactions on control systems technology*, vol. 7, no. 3, pp. 328–342, 1999.

[9] L. Wu, J. Liu, S. Vazquez, and S. K. Mazumder, "Sliding mode control in power converters and drives: A review," *IEEE/CAA Journal of Automatica Sinica*, 2021.

[10] S. Vazquez, J. Rodriguez, M. Rivera, L. G. Franquelo, and M. Norambuena, "Model predictive control for power converters and drives: Advances and trends," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 935–947, 2016.

[11] V. K. Singh, R. N. Tripathi, and T. Hanamoto, "Hil co-simulation of finite set-model predictive control using fpga for a three-phase vsi system," *Energies*, vol. 11, no. 4, p. 909, 2018.

[12] E. Garayalde, I. Aizpuru, U. Iraola, I. Sanz, C. Bernal, and E. Oyarbide, "Finite control set mpc vs continuous control set mpc performance comparison for synchronous buck converter control in energy storage application," in *2019 International Conference on Clean Electrical Power (ICCEP)*, IEEE, 2019, pp. 490–495.

[13] S. K. Gannamraju, D. Valluri, and R. Bhimasingu, "Comparison of fixed switching frequency based optimal switching vector mpc algorithms applied to voltage source inverter for stand-alone applications," in *2019 National Power Electronics Conference (NPEC)*, IEEE, 2019, pp. 1–6.

[14] D. Wang, Z. J. Shen, X. Yin, *et al.*, "Model predictive control using artificial neural network for power converters," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 3689–3699, 2021.

[15] S. Zhao, F. Blaabjerg, and H. Wang, "An overview of artificial intelligence applications for power electronics," *IEEE Transactions on Power Electronics*, vol. 36, no. 4, pp. 4633–4658, 2020.

[16] P. R. Bana, S. Vanti, and M. Amin, "Single-stage grid-connected pv system with artificial neural network controller," in *2021 IEEE 22nd Workshop on Control and Modelling of Power Electronics (COMPEL)*, IEEE, 2021, pp. 1–7.

[17] X. Fu, S. Li, M. Fairbank, D. C. Wunsch, and E. Alonso, "Training recurrent neural networks with the

levenberg–marquardt algorithm for optimal control of a grid-connected converter," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 9, pp. 1900–1912, 2014.

[18] M. Yu, C. Lu, and Y. Liu, "Direct heuristic dynamic programming method for power system stability enhancement," in *2014 American Control Conference*, IEEE, 2014, pp. 747–752.

[19] L. Yang, J. Si, K. S. Tsakalis, and A. A. Rodriguez, "Direct heuristic dynamic programming for nonlinear tracking control with filtered tracking error," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1617–1622, 2009. DOI: 10.1109/TSMCB.2009.2021950.

[20] X. Luo and J. Si, "Stability of direct heuristic dynamic programming for nonlinear tracking control using pid neural network," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–7. DOI: 10.1109/IJCNN.2013.6707054.

[21] J. Xu, Z. Wu, X. Yang, J. Ye, and A. Shen, "Ann-based control method implemented in a voltage source converter for industrial micro-grid," in *2011 Sixth International Conference on Bio-Inspired Computing: Theories and Applications*, IEEE, 2011, pp. 140–145.

[22] N. Malla, U. Tamrakar, D. Shrestha, Z. Ni, and R. Tonkoski, "Online learning control for harmonics reduction based on current controlled voltage source power inverters," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 3, pp. 447–457, 2017.

[23] C. Bajracharya, M. Molinas, J. A. Suul, T. M. Undeland, *et al.*, "Understanding of tuning techniques of converter controllers for vsc-hvdc," in *Nordic Workshop on Power and Industrial Electronics (NORPIE/2008), June 9-11, 2008, Espoo, Finland*, Helsinki University of Technology, 2008.

[24] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural network design*. PWS Publishing Co., 1997.

[25] F.-Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE computational intelligence magazine*, vol. 4, no. 2, pp. 39–47, 2009.

[26] G. Hübner, "Adaptive dynamic programming," *OPTIMIZATION AND OPERATIONS RESEARCH–Volume IV*, p. 119, 2009.

[27] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[28] C. Lu, J. Si, and X. Xie, "Direct heuristic dynamic programming for damping oscillations in a large power system," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 1008–1013, 2008.

Appendix B

Standard Backpropagation Algorithm

```
%% Training of FFNN using standard backpropagation

% Load datasamples
% Normalize into [-1 and 1]
p = 2*(X-min(X))./(max(X)-min(X))-1;
t = 2*(T-min(T))./(max(T)-min(T))-1;

% Number of data samples
[~, samples] = size(p);

% Create a random order of data samples
random_order = randperm(samples);

% Training, validation and testing are assigned a random set of the
% samples
p = p(:,random_order);
t = t(:,random_order);

%% Parametres

% Number of epochs
epochs = 1000;

% Assign a part of the dataset for training, validation and testing
train_ratio = 70/100;   Ntrain  = floor(train_ratio*samples);
val_ratio   = 15/100;   Nval    = floor(val_ratio*samples);
test_ratio  = 15/100;   Ntest   = floor(val_ratio*samples);


% Learning rate
lr = 0.1;

% Validation mean square error
% Chosen high, such that the very first error-increase is sensed
MSE_val_prev = 100;

%% Initialize weights and bias
% Here: Architecture 7-3-3
W1 = 1*rand(3,7)- 0.5;
```

```
b1 = 1*rand(3,1)- 0.5;
W2 = 1*rand(3,3)- 0.5;
b2 = 1*rand(3,1)- 0.5;

W1_prev = zeros(3,7);
b1_prev = zeros(3,1);
W2_prev = zeros(3,3);
b2_prev = zeros(3,1);

%% Iterate through epochs
k = 0;

for epoch = 1:epochs

    %% Training
    sse_train = 0; %Sum Squared Error - SSE
    for i = 1: Ntrain

        % Input Layer
        p_train    = p(:,i);
        t_train   = t(:,i);
        % Hidden layer
        n1_train = W1 * p_train + b1;
        a1_train = tansig_func(n1_train);
        % Output Layer
        n2_train = W2 * a1_train + b2;
        a2_train = purelin(n2_train);
        % Error
        e_train  = t_train - a2_train;
        % Squared error
        ee_train = e_train'*e_train;
        % Sum of squared errors
        sse_train = sse_train + ee_train;

        % Back propagation: Steepest descent
        % w_new = w_old - alpha*df/dw
        % b_new = b_old - alpha*df/db

        % Derivative
        dfdn1 = tansig_der(n1_train);
        dfdn2 = 1;

        % Sensitivity
        s2 = -2*dfdn2*e_train;
        s1 = dfdn1*W2'*s2;

        % Updating
        W2 = W2 - lr*s2* a1_train';
        W1 = W1 - lr*s1* p_train';
        b2 = b2 - lr*s2;
        b1 = b1 - lr*s1;
    end

    % Mean Squared Error
    MSE_train = sse_train/Ntrain;
    MSE_plot_train(epoch) = MSE_train;

    %% Validation
```

```
    sse_val = 0;
    for i = 1:Nval
        p_val  = p(:,i + Ntrain);
        t_val  = t(:,i + Ntrain);

        n1_val = W1 * p_val + b1;
        a1_val = tansig_func(n1_val);
        n2_val = W2 * a1_val + b2;
        a2_val = purelin(n2_val);

        e_val   = t_val - a2_val;
        ee_val  = e_val'*e_val;
        sse_val = sse_val + ee_val;
    end %for validation
    MSE_val = sse_val/Nval;
    MSE_plot_val(epoch) = MSE_val;

    % If error is increasing
    if MSE_val > MSE_val_prev
        k = k+1;
        if k == 1
            W1_val = W1_prev;
            W2_val = W2_prev;
            b1_val = b1_prev;
            b2_val = b2_prev;

            epoch_val = epoch-1;
        end %if k==1

        % Max consequtive validation errors
        if k == 15
            save('network_val', 'W1_val', 'W2_val','b1_val',
↪   'b2_val','epoch_val')
            disp('15 validation fails, the best network is saved.');
            %break
        end % if k==15

    else
        k = 0;
    end %Validation

    MSE_val_prev = MSE_val;
    W1_prev      = W1;
    W2_prev      = W2;
    b1_prev      = b1;
    b2_prev      = b2;

    %% Testing

    sse_test = 0;
    for i = 1: Ntest
        p_test  = p(:, Ntrain + Nval + i);
        t_test  = t(:, Ntrain + Nval + i);

        n1_test = W1 * p_test + b1;
        a1_test = tansig_func(n1_test);
        n2_test = W2 * a1_test + b2;
        a2_test = purelin(n2_test);
```

```
        e_test    =  t_test - a2_test;
        ee_test   = e_test' * e_test;
        sse_test  = sse_test + ee_test;
    end %for test

    % Mean squared error
    MSE_test = sse_test/Ntest;
    MSE_plot_test(epoch) = MSE_test;

end %for epoch

%% Plot
figure(1)
plot(MSE_plot_train); hold on;
plot(MSE_plot_val); hold on;
plot(MSE_plot_test);
ylabel('MSE');
xlabel('Epoch');
grid on;
legend('Training','Validation','Testing');

% Save results
save('Network','W1','W2','b1','b2','epoch');


%% Functions
function g = logsig_func(z)
% Compute log-sigmoid function
    g = 1 ./ (1 + exp(-z));
    i = find(~isfinite(g));
    g(i) = sign(z(i));
end

function g = logsig_der(z)
% Compute derivative of log-sigmoid function
d = zeros(size(z));
 for i = length(z)
     d(i)=(1-logsig_func(z(i)))*logsig_func(z(i));
 end
 g = diag(d);
end

function a = tansig_func(z)
% Compute tan-sigmoid function
a = 2./(1+exp(-2*z))-1;
end

function dt = tansig_der(z)
% Compute tan-sigmoid derivative
d = zeros(size(z));
for i = length(z)
    d(i) = 1-(tansig(z(i)))^2;
end
dt = diag(d);
end
```

Appendix C

Levenberg-Marquardt Algorithm

```
%% Training ANN
% using embedded functions

% Load data
x = X;
t = T;

% Training algorithm
trainFcn            = 'trainlm';

% traingd - gradient descent
% net.trainParam.lr   = 0.05;

% Architecture
hidden_neurons = 3; %for more layers: [x y]
net = feedforwardnet(hidden_neurons, trainFcn);

% Transfer function
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'purelin';

% Separate data between Training, Validation, Testing
net.divideParam.trainRatio  = 70/100;
net.divideParam.valRatio    = 15/100;
net.divideParam.testRatio   = 15/100;

% Number of epochs
net.trainParam.epochs       = 1000;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
```

```
% View Network
view(net)

% Generate Simulink block
gensim(net)

% Plot Performance
figure(2)
plot(tr.perf);  hold on;
plot(tr.vperf); hold on;
plot(tr.tperf); hold on;
ylabel('MSE');
xlabel('Epoch');
legend('training','validation','testing');
xlim([1,net.trainParam.epochs]);
```

Appendix D

Forward Accumulation Through Time

D.1 Prepare data

```
%% Prepare data for RNN-training
% Store data using 'to-file' block in Simulink
% Store data as timeseries, using sampling time T=1e-4s
% Input X consists of 4 variables: ed,eq,sd,sq

clear;
X = load('input.mat'); X = (X.ans.Data)';
i_ref = load('target.mat'); i_ref = (i_ref.ans.Data)';
```

D.2 Training Algorithm

```
%% FATT + LM Algorithm
% 1. Retrieve data
% 2. Define system and controller parameters
% 3. Initialize
% 4. Train network according to LM+FATT

%% 1. Retrieve Data
x = X;
iref = i_ref;
[~,samples] = size(X);

%% 2. Parameters

% System parameters
Ts = 1e-4;      % Sampling time
Lf = 1.55e-3;   % Filter inductance
Rf = 0.01;      % Filter resistance
fn = 50;        % Nominal grid frequency
wn = 2*pi*fn;   % Nominal angular speed
Kpwm = 500/2;   % PWM-gain
vg =[280;0];    % Grid voltage

% Discrete model
A = [1-Rf*Ts/Lf Ts*wn; -Ts*wn 1-Rf*Ts/Lf];
```

```
B = Ts/Lf;

% Control parameters
epochs = 100;
mu = 0.001;
mu_inc = 10;
mu_dec = 0.1;
dCdw_min = 1e-10;
alpha = 0.2;

%% 3. Initialize

% LM+FATT variables
C_epoch = zeros(epochs,1);
C = 0; % Cost-to-go
J = zeros(samples,72);
U = zeros(1,samples);
V = zeros(samples,1);

% Random weight elements in the range [-0.1, 0.1]
W1 = 0.2.*rand(6,4)-0.1; % 4 inputs: e_dq, s_dq
W2 = 0.2.*rand(6,6)-0.1; % 2 hidden layers: 6 neurons
W3 = 0.2-*rand(2,6)-0.1;% 2 outputs: v_cdq*

% Convert from matrix to vector
w1_vec = reshape(W1',[1,6*4]);  % (1x24)
w2_vec = reshape(W2',[1,6*6]);  % (1x36)
w3_vec = reshape(W3',[1,2*6]);  % (1x12)
w_vec =[w1_vec, w2_vec, w3_vec];% (1x72)

% Variables
i = zeros(2,samples);
u = zeros(2,samples);
e = zeros(2,samples);
s = zeros(2,samples);

% Derivatives
dsdw = cell([1,samples]);
didw = cell([1,samples]);
dudw = cell([1,samples]);
dphidw = cell([1,samples]);
dVde = zeros(samples,2);
dVdw = cell([1,samples]);
for sample =1:samples
    dsdw{sample} = zeros(2,72);
    didw{sample} = zeros(2,72);
    dudw{sample} = zeros(2,72);
    dphidw {sample} = zeros(2,72);
    dVdw{sample} = zeros(1,72);
end

%% 4. Training
for epoch = 1:epochs
     for k=1:samples-1
        % Retrieve data, pass through network
        p = x(:,k);                                  % get inputs
        [a3,n3,a2,n2,a1,n1,a] = RNN(p,W1,W2,W3);     % get outputs
        u(:,k) = Kpwm*a3 - vg;                       % control signal
```

```
    % Derivatives
    [dadn3,dadn2,dadn1,dadn] = dadn_func(a3,n3,a2,n2,a1,n1,a,p);
    [dAde, dAds] = dAdp_func(W3,W2,W1,dadn3,dadn2,dadn1,dadn);
    [dndw3, dndw2, dndw1] = dndw_func(a2,a1,a);
    [dAdw] = dAdw_func(dadn3,dndw3,W3,dadn2,dndw2,W2,dadn1,dndw1);

    dsdw{k} = Ts * (dphidw{k} - (1/2)*didw{k});
    dudw{k} = Kpwm * (dAde*didw{k} + dAds*dsdw{k} + dAdw);
    didw{k+1} = A * didw{k} + B * dudw{k};

    i(:,k+1) = A*i(:,k) + B*u(:,k);
    e(:,k+1) = i(:,k+1) - iref(:,k+1);
    s(:,k+1) = s(:,k) + (Ts/2)*(e(:,k+1) - e(:,k));
    V(k) = (e(1,k+1)^2 + e(2,k+1)^2)^(alpha/2);

    C = C + V(k)^2; %DP cost

    dVde(k+1,:) = alpha*(e(1,k+1)^2 ...
        + e(2,k+1)^2)^((alpha/2)-1 ) * [e(1,k+1), e(2,k+1)];
    dphidw{k+1} = dphidw{k} + didw{k+1};
    dVdw{k+1} = dVde(k+1,:)*(didw{k+1});
    J(k+1,:) = dVdw{k+1}; %Jacobian
end

dCdw = 2*J'*V; %Gradient
if norm(dCdw) >= dCdw_min
    dw = -((J'*J) + mu*eye(size(J'*J)))\(J'*V);
    w_vec_new = w_vec + dw;
    W1_new = (reshape(w_vec_new(1:24),[4,6]))';
    W2_new = (reshape(w_vec_new(24+1:24+36),[6,6]))';
    W3_new = (reshape(w_vec_new(24+36+1:24+36+12),[6,2]))';

    for k = 1:samples-1
         p = x(:,k);
         [a3,n3,a2,n2,a1,n1,a] = RNN(p,W1_new,W2_new,W3_new);
         u(:,k) = Kpwm*a3 - vg;
         i(:,k+1) = A*i(:,k) + B*u(:,k);
         e(:,k+1) = i(:,k+1) - iref(:,k+1);
         s(:,k+1) = s(:,k) + (Ts/2)*(e(:,k+1) - e(:,k));
         V(k) = (e(1,k+1)^2 + e(2,k+1)^2)^(alpha/2);
         C_new = C + V(k)^2;
    end
    if C_new < C % If cost is decreased
        C = C_new;
        W1 = W1_new;
        W2 = W2_new;
        W3 = W3_new;
        w_vec = w_vec_new;
        mu = mu*mu_dec; %Decrease mu

    else % If cost has increased
        mu = mu*mu_inc; %Increase mu
        dw = (J'*J + mu*eye(size(J'*J)))\(J'*V);
        w_vec_new = w_vec + dw;
        W1_new = (reshape(w_vec_new(1:24),[4,6]))';
        W2_new = (reshape(w_vec_new(24+1:24+36),[6,6]))';
        W3_new = (reshape(w_vec_new(24+36+1:24+36+12),[6,2]))';
```

```
            for k = 1:samples-1
                 p = x(:,k);
                 [a3,n3,a2,n2,a1,n1,a] = RNN(p,W1_new,W2_new,W3_new);
                 u(:,k) = Kpwm*a3 - vg;
                 i(:,k+1) = A*i(:,k) + B*u(:,k);
                 e(:,k+1) = i(:,k+1) - iref(:,k+1);
                 s(:,k+1) = s(:,k) + (Ts/2)*(e(:,k+1) - e(:,k));
                 V(k) = (e(1,k+1)^2 + e(2,k+1)^2)^(alpha/2) ;
                 C_new = C + V(k)^2;
            end %for
        end %if C
    else
        fprintf('Stop training\n');
        fprintf('epoch = %d\n',epoch);
        break;
    end %if dC/dw
    C_epoch(epoch) = C;
end


%% Functions for calculating
% Network output: RNN()
% Network derivatives: dadn_func(), dAdp_func(),dndw_func(),dAdw_func()

function [a3,n3,a2,n2,a1,n1,a] = RNN(p,W1, W2,W3)
a = tansig(p);
n1 = W1*a;
a1 = tansig(n1);
n2 = W2*a1;
a2 = tansig(n2);
n3 = W3*a2;
a3 = tansig(n3);
end
function [dadn_3,dadn_2,dadn_1,dadn] = dadn_func(a3,n3,a2,n2,a1,n1,a,p)
dadn_3 = diag(dtansig(n3,a3));
dadn_2 = diag(dtansig(n2,a2));
dadn_1 = diag(dtansig(n1,a1));
dadn = diag(dtansig(p,a));
end
function [dAde, dAds] = dAdp_func(W3,W2,W1,dadn3,dadn2,dadn1,dadn)
dAde =  dadn3*W3 * dadn2*W2 * dadn1*W1(:,1:2) * dadn(1:2,1:2); % (2x2)
dAds =  dadn3*W3 * dadn2*W2 * dadn1*W1(:,3:4) * dadn(3:4,3:4); % (2x2)
end
function [dndw_3, dndw_2, dndw_1] = dndw_func(a2,a1,a)

% W3 = 2x6, W3_vec=1x12 = w_61 ... w72
% n3 = 2x1
% a2 = 6x1
%             _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
% n3 =       |w_61*a_1 + w_62*a_2 + ... +w_66*a_6|
%            |w_67*a_1 + w_68*a_2 + ... +w_72*a_6|
%             - - - - - - - - - - - - - - - - - - - -
%             _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
% dndw_3 =  |dn_1/dw_61  dn_1/dw_62 ... dn_1/dw_72|
%           |dn_2/dw_61  dn_2/dw_62 ... dn_2/dw_72|
%            - - - - - - - - - - - - - - - - - - - -
%             _ _ _ _ _ _ _ _ _ _ _ _
% dndw_3 =  |a_1 ... a_6  0  ...   0|
```

```
%           | 0  ...  0  a_1 ... a_6|
%            - - - - - - - - - - - -

dndw_3 = [a2' zeros(1,6); zeros(1,6) a2'];
dndw_2 = zeros(6,36);
dndw_1 = zeros(6,24);

i2=1; i3=1;
for i=1:length(a1)
    dndw_2(i,i2:i2+5) = a1';
    dndw_1(i,i3:i3+3) = a';
    i2=i*6+1;
    i3=i*4+1;
end
end
function [dAdw] = dAdw_func(dadn3,dndw3,W3,dadn2,dndw2,W2,dadn1,dndw1)
dAdw3 = dadn3 * dndw3;
dAdw2 = dadn3 * W3 * dadn2 * dndw2;
dAdw1 = dadn3*W3 * dadn2 * W2*dadn1 * dndw1;
dAdw = [dAdw1 dAdw2 dAdw3];
end
```

Appendix E

Continuous Control Set Model Predictive Control

E.1 System Matrices

```
% Parameters
Cf  = 75e-6; Lf  = 1.55e-3; Rf  = 0.01;
Ts  = 1e-6; N = 4;

% Continuous: dxdt= A x(t) + B v_conv(t) Bd v_grid(t)
Ac = [-Rf/Lf 0; 0 -Rf/Lf];
Bc = [1/Lf 0; 0 1/Lf];
Bcd = -Bc;
nx = size(Ac,2);

% Discrete: x[k+1] = Ad x[k] + Bd v_conv[k] + Bdd v_grid[k]
Ad  = expm(Ac*Ts);
Bd  = (Ad - eye(length(Ad)))*Ac^(-1)*Bc;
Bdd = (Ad - eye(length(Ad)))*Ac^(-1)*Bcd;

% Single Horizon:
% Phi=Ad;
% Gamma = Bd;
% Gammad = Bdd;

% Extend Horizon
A1=Ad;      A2=Ad^2 ;
A3=Ad^3;    A4=Ad^4 ;

Phi   = [Ad ; Ad^2 ; Ad^3 ; Ad^4];
Gamma = [Bd                  zeros(2,6);
         Ad*Bd  Bd           zeros(2,4);
         Ad^2*Bd  Ad*Bd Bd   zeros(2,2);
         Ad^3*Bd   Ad^2*Bd   Ad*Bd   Bd];

Gammad = [Bdd zeros(2,6);
         Ad*Bdd Bdd zeros(2,4);
         Ad^2*Bdd Ad*Bdd Bdd zeros(2,2);
         Ad^3*Bdd Ad^2*Bdd Ad*Bdd Bdd];
```

```
W=diag(ones(N,1));
```

E.2 CCS-MPC Algorithm

```
function Vabc_conv  = pred_ctrl(Iab_ref, Vab_grid, Iab_conv, Vdc_ref, Phi, Gamma,
↪  Gammad, Ts)

persistent w0 T_clarke W N A

if isempty(T_clarke)
    N = 4;
    w0=2*pi*50;
    T_clarke = 2/3*[1 -1/2 -1/2;0 sqrt(3)/2 -sqrt(3)/2];
    W = diag(ones(2*N,1));
    A = inv(Gamma'*W*Gamma);
end

Vgrid_k      = zeros(2*N,1);
Vgrid_k(1)   = Vab_grid(1);
Vgrid_k(2)   = Vab_grid(2);

Iref_k      = zeros(2*N,1);
Iref_k(1)   = Iab_ref(1);
Iref_k(2)   = Iab_ref(2);



% Predict
Vgrid_k1 = Vgrid_k;
Iref_k1  = Iref_k;

for i=1:N-1
    index=i*2+1;
    indexp=(i-1)*2+1;

    % Predict Grid Voltage
    Vgrid_k1((index):(index+1),1)=...
        [Vgrid_k1((indexp),1)-Vgrid_k1((indexp+1),1)*w0*Ts;
        Vgrid_k1(indexp+1,1)+Vgrid_k1(indexp,1)*w0*Ts];

    % Predict Reference Current
    Iref_k1((index):(index+1),1)=...
        [Iref_k1((indexp),1)-Iref_k1(indexp+1,1)*w0*Ts;
        Iref_k1(indexp+1,1)+Iref_k1(indexp,1)*w0*Ts];
end

% Optimal converter voltage
B = Gamma'*W* Iref_k1 - Gamma'*W*Gammad* Vgrid_k1 - Gamma'*W*Phi* Iab_conv;
Vab_conv = A*B;
Vabc_conv = T_clarke'*Vab_conv(1:2,1)/(Vdc_ref/2);
```

Appendix F

Direct Heuristic Dynamic Programming Algorithm

F.1 Offline Training

```
% Initialize Weights
W1_a = (0.1--0.1)*rand(12,6)+(-0.1); % 6 inputs, 12 hidden
W2_a = 0.2*rand(2,12)- 0.1;          % 12 hidden, 2 outputs
W1_c = 0.2*rand(12,8)-0.1;           % 8 inputs, 12 hidden
W2_c = 0.2*rand(1,12)-0.1;           % 12 hidden, 1 output

W1_a_init = W1_a;
W2_a_init = W2_a;
W1_c_init = W1_c;
W2_c_init = W2_c;

% Initialize training Parameters
lr = 0.01;
alpha = 0.1;
Nc = 50; Tc = 1e-7;
Na = 40; Ta = 1e-6;
c1 = 0.4; c2 = 0.2; c3 =0.04;
Nhidden = 12;

% Get samples
[~, Nsamples] = size(X);
T = Nsamples;
X_norm = (2*(X-min(X))./(max(X)-min(X)))-1; %normalize X [-1,+1]

% Initialize variables
u = zeros(2,T);
J = zeros(1,T);          r = zeros(1,T);
ea = zeros(1,T);         ec = zeros(1,T);
Ea = ones(1,T);          Ec = ones(1,T);
Ea_k = zeros(1,Nc);      Ec_k = zeros(1,Na);
temp_d = 0;              temp_q = 0;
dEdW1_c = zeros(8,12);   dEdW1_a = zeros(6,12);
dEdW2_c = zeros(12,1);   dEdW2_a = zeros(12,2);
```

```
%% Training

for t=1:1:T % Repeat (for each step t of trial):
    kc=0; ka =0;
    x = X_norm(:,t);
    [u(:,t), a1_a] = nn_a(W1_a, W2_a, x);
    xu = [x ;u(:,t)];
    [J(t), a1_c] = nn_c(W1_c, W2_c, xu);
    r(t) = (c1*x(1) + c2*x(2) + c3*x(3) + c1*x(4) + c2*x(5) + c3*x(6));
    if t>=2

    % Critic update
    while (kc <= Nc) && (Ec(t) >= Tc)
        [J(t), a1_c] = nn_c(W1_c, W2_c, xu);
        kc = kc + 1;
        ec(t) = alpha*J(t) - J(t-1) + r(t);
        Ec(t) = 0.5 *ec(t)^2;
        Ec_k(kc) = Ec(t);
        % Derivative dEdW_c
        for k = 1:Nhidden
            dEdW2_c(k) = ec(t) * alpha * a1_c(k);
            for i = 1:length(xu)
                dEdW1_c(i,k) = ec(t) * alpha * W2_c(k)*(1-a1_c(k)^2)*xu(i);
            end
        end
        W1_c = W1_c + lr * (dEdW1_c');
        W2_c = W2_c + lr * (dEdW2_c');
    end

    % Action update
    while (ka <= Na) && (Ea(t) >= Ta)
        ka = ka + 1;
        [u(:,t), a1_a] = nn_a(W1_a, W2_a, x);
        xu = [x; u(:,t)];
        [J(t), a1_c] = nn_c(W1_c, W2_c, xu);
        ea(t) = J(t) ;
        Ea(t) = 0.5 *ea(t)^2;
        Ea_k(ka) = Ea(t);
        % Derivative dEdW_a
        for j = 1:Nhidden
            for i =1: length(x)
                for k = 1:Nhidden
                    temp_d =  temp_d +  ea(t)*( W2_c(k) * (1-a1_c(k)^2) *
↪   W1_c(k,7) );
                    temp_q =  temp_d +  ea(t)*( W2_c(k) * (1-a1_c(k)^2) *
↪   W1_c(k,8) );
                end
                dEdW1_a(i,j) = (temp_d+temp_q)*(1-(a1_a(j))^2)*x(i);
            end % for i = x
            temp_d = temp_d*a1_a(j);
            temp_q = temp_q*a1_a(j);
            dEdW2_a(j,:) = [temp_d temp_q];
        end % for j = hidden
        W1_a = W1_a + lr *(dEdW1_a');
        W2_a = W2_a + lr *(dEdW2_a');
    end % while
    end % if t>=2
end %for t
```

```
function [u,a1_a] = nn_a(W1_a, W2_a, x)
    n1_a = (W1_a * x);
    a1_a = tansig(n1_a);
    u = W2_a * a1_a;
    u = 1*u(:);
end

function [J,a1_c] = nn_c(W1_c,W2_c, xu)
    n1_c = W1_c * xu;
    a1_c = tansig(n1_c);
    J = W2_c * a1_c;
end
```

F.2 Online Operation

```
function [ud,uq] = ADP(W1a,W2a,W1c,W2c,ed_t, ed_t1, ed_t2, eq_t, eq_t1, eq_t2)

% tansig faster than hyperbolic tangent (but mathematically equivalent)
coder.extrinsic('tansig');


% Training parameters
persistent T_duration T_step T Nhidden lr alpha Nc Na Tc Ta c1 c2 c3
if isempty(T_duration)
    T_duration = 0.5; % Simulation time
    T_step = 1e-6; % Sampling time
    T = floor(T_duration/T_step)+1;
    Nhidden = 12;
    lr = 0.001;
    alpha = 0.1;
    Nc = 20; Tc = 1e-7;
    Na = 10; Ta = 1e-6;
    c1 = 0.4; c2 = 0.2; c3 =0.04;
end

% Outputs/Inputs: Allocate
persistent t u r J J_old
if isempty(t)
    t = 1;
    J = 0;
    J_old = 0;
end

%% Errors and derivatives: Allocate
persistent dEdW1_a dEdW2_a dEdW1_c dEdW2_c ec ea Ea Ec temp_d temp_q
if isempty(dEdW1_a)
    dEdW1_a = zeros(6,12); dEdW1_c = zeros(8,12);
    dEdW2_a = zeros(12,2); dEdW2_c = zeros(12,1);
end

%% Weights obtained from offline training
persistent W1_a W2_a W1_c W2_c a1_a a1_c n1_a n1_c
if isempty(W1_a)
    W1_a = W1a;
    W2_a = W2a;
```

```
    W1_c = W1c;
    W2_c = W2c';
    a1_a = zeros(12,1);
    a1_c = zeros(12,1);

end

%% Repeat (for each trial):
% Initialize: X(t) = inputs
x = [ed_t; ed_t1; ed_t2; eq_t; eq_t1; eq_t2];
x = 2*(x-min(x))./(max(x)-min(x))-1; %[-1,+1]
Ec = 1; Ea = 1;
temp_d = 0; temp_q=0;

% Actor, action : u(t) = nn_a(X(t))
n1_a = W1_a * x(:);
a1_a = tansig(n1_a);
u = W2_a * a1_a;

% Critic, cost-to-go: J(t) = nn_c(X(t),u(t))
xu = [x; u];
n1_c = W1_c *xu;
a1_c = tansig(n1_c);
J = W2_c *a1_c;

% Take action u(t), observe r(t), X(t+1)
ud = u(1);
uq = u(2);
r = c1*x(1) + c2*x(2) + c3*x(3) + c1*x(4) + c2*x(5) + c3*x(6);
kc = 0; ka = 0;
if t>=2
    while (kc <= Nc) && (Ec >= Tc)
        n1_c = W1_c *xu;
        a1_c = tansig(n1_c);
        J_old =J;
        J = W2_c *a1_c;
        ec = alpha*J - (J_old-r);
        Ec = 0.5 *ec^2;
        % Derivative dEdw_c
        for i = 1:Nhidden
            dEdW2_c(i) = ec * alpha * a1_c(i);
            for j = 1:8
                dEdW1_c(j,i) = ec* alpha * W2_c(i)*(1-a1_c(i)^2)*xu(j);
            end
        end
        W1_c = W1_c + lr * (dEdW1_c');
        W2_c = W2_c + lr * (dEdW2_c');
        kc = kc + 1;
    end % Critic-loop

    while (ka <= Na) && (Ea >= Ta)
        n1_a = W1_a * x;
        a1_a = tansig(n1_a);
        u    = W2_a * a1_a;
        xu = [x; u];
        n1_c = W1_c *xu;
        a1_c = tansig(n1_c);
        J = W2_c *a1_c;
```

```
        ea = J;
        Ea = 0.5*ea^2;
        % Derivative dEdW_a
        for j =1:Nhidden
            for i = 1:length(x)
                for k = 1:Nhidden
                    temp_d =  temp_d +  ea*( W2_c(k) * (1-a1_c(k)^2) * W1_c(k,7)
↪   );
                    temp_q =  temp_d +  ea*( W2_c(k) * (1-a1_c(k)^2) * W1_c(k,8)
↪   );
                end
                dEdW1_a(i,j) = (temp_d+temp_q)*(1-(a1_a(j))^2)*x(i);
            end
            temp_d = temp_d*a1_a(j);
            temp_q = temp_q*a1_a(j);
            dEdW2_a(j,:) = [temp_d temp_q];
        end
        W1_a = W1_a + lr *(dEdW1_a');
        W2_a = W2_a + lr *(dEdW2_a');
        ka = ka + 1;
    end
end
t = t+1;
```

Bibliography

[1] IEA, 'Global energy review 2021', *Global Energy Review*, Apr. 2021.

[2] S. D'Arco, *Grid forming*, ELK-23, Power Electronics in Future Power Systems, Lecture 9, NTNU, Department of Electric Power Engineering, 5th Nov. 2021.

[3] R. D. Pereira, M. Veronesi, A. Visioli, J. E. Normey-Rico and B. C. Torrico, 'Implementation and test of a new autotuning method for pid controllers of tito processes', *Control Engineering Practice*, vol. 58, pp. 171–185, 2017.

[4] J. Liu, Y. Yin, W. Luo, S. Vazquez, L. G. Franquelo and L. Wu, 'Sliding mode control of a three-phase ac/dc voltage source converter under unknown load conditions: Industry applications', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 10, pp. 1771–1780, 2017.

[5] Y. Gui, X. Wang, F. Blaabjerg and D. Pan, 'Control of grid-connected voltage-source converters: The relationship between direct-power control and vector-current control', *IEEE Industrial Electronics Magazine*, vol. 13, no. 2, pp. 31–40, 2019.

[6] I. S. Mohamed, S. Rovetta, T. D. Do, T. Dragicević and A. A. Z. Diab, 'A neural-network-based model predictive control of three-phase inverter with an output *LC* filter', *IEEE Access*, vol. 7, pp. 124 737–124 749, 2019.

[7] L. Wu, J. Liu, S. Vazquez and S. K. Mazumder, 'Sliding mode control in power converters and drives: A review', *IEEE/CAA Journal of Automatica Sinica*, 2021.

[8] S. Vazquez, J. Rodriguez, M. Rivera, L. G. Franquelo and M. Norambuena, 'Model predictive control for power converters and drives: Advances and trends', *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 935–947, 2016.

[9] V. K. Singh, R. N. Tripathi and T. Hanamoto, 'Hil co-simulation of finite set-model predictive control using fpga for a three-phase vsi system', *Energies*, vol. 11, no. 4, p. 909, 2018.

[10] S. Zhao, F. Blaabjerg and H. Wang, 'An overview of artificial intelligence applications for power electronics', *IEEE Transactions on Power Electronics*, vol. 36, no. 4, pp. 4633–4658, 2020.

[11] Y. Gao and Y.-J. Liu, 'Adaptive fuzzy optimal control using direct heuristic dynamic programming for chaotic discrete-time system', *Journal of Vibration and Control*, vol. 22, no. 2, pp. 595–603, 2016.

[12] P. R. Bana, S. Vanti and M. Amin, 'Single-stage grid-connected pv system with artificial neural network controller', in *2021 IEEE 22nd Workshop on Control and Modelling of Power Electronics (COMPEL)*, IEEE, 2021, pp. 1–7.

[13] X. Fu and S. Li, 'Control of single-phase grid-connected converters with lcl filters using recurrent neural network and conventional control methods', *IEEE Transactions on Power Electronics*, vol. 31, no. 7, pp. 5354–5364, 2015.

[14] P. R. Bana and M. Amin, 'Adaptive vector control of grid-tied vsc using multilayer perceptron-recurrent neural network', in *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2021, pp. 1–6.

[15] N. Malla, U. Tamrakar, D. Shrestha, Z. Ni and R. Tonkoski, 'Online learning control for harmonics reduction based on current controlled voltage source power inverters', *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 3, pp. 447–457, 2017.

[16] M. Yu, C. Lu and Y. Liu, 'Direct heuristic dynamic programming method for power system stability enhancement', in *2014 American Control Conference*, IEEE, 2014, pp. 747–752.

[17] L. Yang, J. Si, K. S. Tsakalis and A. A. Rodriguez, 'Direct heuristic dynamic programming for nonlinear tracking control with filtered tracking error', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1617–1622, 2009. DOI: 10.1109/TSMCB.2009.2021950.

[18] X. Luo and J. Si, 'Stability of direct heuristic dynamic programming for nonlinear tracking control using pid neural network', in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–7. DOI: 10.1109/IJCNN.2013.6707054.

[19] A. De Oliveira, C. Tiburcio, M. Lemes and D. Retzmann, 'Prospects of voltage-sourced converters (vsc) applications in dc transmission systems', in *2010 IEEE/PES Transmission and Distribution Conference and Exposition: Latin America (T&D-LA)*, IEEE, 2010, pp. 491–495.

[20] ENTSO-E. 'Voltage source converters'. (), [Online]. Available: https://www.entsoe.eu/Technopedia/techsheets/voltage-source-converters (visited on 6th Dec. 2021).

[21] K. Li and C. Zhao, 'New technologies of modular multilevel converter for vsc-hvdc application', in *2010 Asia-Pacific Power and Energy Engineering Conference*, IEEE, 2010, pp. 1–4.

[22] K. K. Sen and M. L. Sen, *Introduction to FACTS controllers: theory, modeling, and applications*. John Wiley & Sons, 2009, vol. 54.

[23] K. Zeb *et al.*, 'A comprehensive review on inverter topologies and control strategies for grid connected photovoltaic system', *Renewable and Sustainable Energy Reviews*, vol. 94, pp. 1120–1141, 2018.

[24] M. Amin, 'Small-signal stability characterization of interaction phenomena between hvdc system and wind farms', 2017.

[25] P. C. Krause, O. Wasynczuk, S. D. Sudhoff and S. D. Pekarek, *Analysis of electric machinery and drive systems*. John Wiley & Sons, 2013, vol. 75.

[26] R. H. Park, 'Two-reaction theory of synchronous machines generalized method of analysis-part i', *Transactions of the American Institute of Electrical Engineers*, vol. 48, no. 3, pp. 716–727, 1929.

[27] P. C. Krause and C. Thomas, 'Simulation of symmetrical induction machinery', *IEEE transactions on power apparatus and systems*, vol. 84, no. 11, pp. 1038–1053, 1965.

[28] A. Iqbal, E. Levi, M. Jones and S. N. Vukosavic, 'Generalised sinusoidal pwm with harmonic injection for multi-phase vsis', in *2006 37th IEEE Power Electronics Specialists Conference*, IEEE, 2006, pp. 1–7.

[29] L. Malesani and P. Tomasin, 'Pwm current control techniques of voltage source converters-a survey', in *Proceedings of IECON'93-19th Annual Conference of IEEE Industrial Electronics*, IEEE, 1993, pp. 670–675.

[30] D. G. Holmes and T. A. Lipo, *Pulse width modulation for power converters: principles and practice*. John Wiley & Sons, 2003, vol. 18.

[31] T. Midtsund, 'Control of power electronic converters in distributed power generation systems: Evaluation of current control structures for voltage source converters operating under weak grid conditions', M.S. thesis, Institutt for elkraftteknikk, 2010.

[32] N. Jaalam, N. Rahim, A. Bakar, C. Tan and A. M. Haidar, 'A comprehensive review of synchronization methods for grid-connected converters of renewable energy source', *Renewable and Sustainable Energy Reviews*, vol. 59, pp. 1471–1481, 2016.

[33] M. P. Kazmierkowski and L. Malesani, 'Current control techniques for three-phase voltage-source pwm converters: A survey', *IEEE Transactions on industrial electronics*, vol. 45, no. 5, pp. 691–703, 1998.

[34] F. Blaabjerg, R. Teodorescu, M. Liserre and A. V. Timbus, 'Overview of control and grid synchronization for distributed power generation systems', *IEEE Transactions on industrial electronics*, vol. 53, no. 5, pp. 1398–1409, 2006.

[35] A. Egea-Alvarez, S. Fekriasl, F. Hassan and O. Gomis-Bellmunt, 'Advanced vector control for voltage source converters connected to weak grids', *IEEE Transactions on Power Systems*, vol. 30, no. 6, pp. 3072–3081, 2015.

[36] S. Fukuda and T. Yoda, 'A novel current-tracking method for active filters based on a sinusoidal internal model [for pwm invertors]', *IEEE Transactions on Industry Applications*, vol. 37, no. 3, pp. 888–895, 2001.

[37] H. Cha, T.-K. Vu and J.-E. Kim, 'Design and control of proportional-resonant controller based photovoltaic power conditioning system', in *2009 IEEE Energy Conversion Congress and Exposition*, IEEE, 2009, pp. 2198–2205.

[38] P. Rioual, H. Pouliquen and J. Louis, 'Non linear control of pwm rectifier by state feedback linearization and exact pwm control', in *Proceedings of 1994 Power Electronics Specialist Conference-PESC'94*, IEEE, vol. 2, 1994, pp. 1095–1102.

[39] E. Song, A. F. Lynch and V. Dinavahi, 'Experimental validation of nonlinear control for a voltage source converter', *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1135–1144, 2009.

[40] R. M. Milasi, 'Adaptive and nonlinear control of a voltage source converter', Ph.D. dissertation, University of Alberta, 2012.

[41] K. D. Young, V. I. Utkin and U. Ozguner, 'A control engineer's guide to sliding mode control', *IEEE transactions on control systems technology*, vol. 7, no. 3, pp. 328–342, 1999.

[42] T. Kawabata, T. Miyashita and Y. Yamamoto, 'Dead beat control of three phase pwm inverter', in *1987 IEEE Power Electronics Specialists Conference*, IEEE, 1987, pp. 473–481.

[43] K. P. Gokhale, A. Kawamura and R. G. Hoft, 'Dead beat microprocessor control of pwm inverter for sinusoidal output waveform synthesis', *IEEE Transactions on Industry Applications*, no. 5, pp. 901–910, 1987.

[44] E. Garayalde, I. Aizpuru, U. Iraola, I. Sanz, C. Bernal and E. Oyarbide, 'Finite control set mpc vs continuous control set mpc performance comparison for synchronous buck converter control in energy storage application', in *2019 International Conference on Clean Electrical Power (ICCEP)*, IEEE, 2019, pp. 490–495.

[45] L. A. Zadeh, 'Fuzzy logic', *Computer*, vol. 21, no. 4, pp. 83–93, 1988.

[46] B. K. Bose, 'Artificial intelligence techniques in smart grid and renewable energy systems—some example applications', *Proceedings of the IEEE*, vol. 105, no. 11, pp. 2262–2273, 2017.

[47] U. Yilmaz, A. Kircay and S. Borekci, 'Pv system fuzzy logic mppt method and pi control as a charge controller', *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 994–1001, 2018.

[48] H. M. Hasanien and M. Matar, 'A fuzzy logic controller for autonomous operation of a voltage source converter-based distributed generation system', *IEEE Transactions on Smart grid*, vol. 6, no. 1, pp. 158–165, 2014.

[49] M. Rosyadi, S. Muyeen, R. Takahashi and J. Tamura, 'Fuzzy logic controlled voltage source converter in grid connected application via lcl filter', in *2012 15th International Conference on Electrical Machines and Systems (ICEMS)*, IEEE, 2012, pp. 1–6.

[50] A. I. Al-Odienat and A. A. Al-Lawama, 'The advantages of pid fuzzy controllers over the conventional types', *American Journal of Applied Sciences*, vol. 5, no. 6, pp. 653–658, 2008.

[51] S. Zhao, F. Blaabjerg and H. Wang, 'An overview of artificial intelligence applications for power electronics', *IEEE Transactions on Power Electronics*, 2020.

[52] V. S. B. Kurukuru, A. Haque, M. A. Khan, S. Sahoo, A. Malik and F. Blaabjerg, 'A review on artificial intelligence applications for grid-connected solar photovoltaic systems', *Energies*, vol. 14, no. 15, p. 4690, 2021.

[53] D. Wang *et al.*, 'Model predictive control using artificial neural network for power converters', *IEEE Transactions on Industrial Electronics*, 2021.

[54] X. Fu, S. Li, M. Fairbank, D. C. Wunsch and E. Alonso, 'Training recurrent neural networks with the levenberg–marquardt algorithm for optimal control of a grid-connected converter', *IEEE transactions on neural networks and learning systems*, vol. 26, no. 9, pp. 1900–1912, 2014.

[55] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators', *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[56] H.-C. Chan, K. Chau and C. Chan, 'A neural network controller for switching power converters', in *Proceedings of IEEE Power Electronics Specialist Conference-PESC'93*, IEEE, 1993, pp. 887–892.

[57] M. Novak and F. Blaabjerg, 'Supervised imitation learning of fs-mpc algorithm for multilevel converters', in *2021 23rd European Conference on Power Electronics and Applications (EPE'21 ECCE Europe)*, IEEE, 2021, P–1.

[58] D. S. Sarali, V. A. I. Selvi and K. Pandiyan, 'An improved design for neural-network-based model predictive control of three-phase inverters', in *2019 IEEE International Conference on Clean Energy and Energy Efficient Electronics Circuit for Sustainable Development (INCCES)*, IEEE, 2019, pp. 1–5.

[59] T. W. Chow and Y. Fang, 'A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics', *IEEE transactions on industrial electronics*, vol. 45, no. 1, pp. 151–161, 1998.

[60] G. Capizzi, F. Bonanno and C. Napoli, 'Recurrent neural network-based control strategy for battery energy storage in generation systems with intermittent renewable energy sources', in *2011 International Conference on Clean Electrical Power (ICCEP)*, IEEE, 2011, pp. 336–340.

[61] F. Bonassi and R. Scattolini, 'Recurrent neural network-based internal model control design for stable nonlinear systems', *European Journal of Control*, vol. 65, p. 100 632, 2022.

[62] S. Sabzevari, R. Heydari, M. Mohiti, M. Savaghebi and J. Rodriguez, 'Model-free neural network-based predictive control for robust operation of power converters', *Energies*, vol. 14, no. 8, p. 2325, 2021.

[63] M. T. Hagan, H. B. Demuth and M. Beale, *Neural network design*. PWS Publishing Co., 1997.

[64] I. Hammoud, S. Hentzelt, T. Oehlschlaegel and R. Kennel, 'Long-horizon direct model predictive control based on neural networks for electrical drives', in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2020, pp. 3057–3064.

[65] S. Wang, T. Dragicevic, Y. Gao and R. Teodorescu, 'Neural network based model predictive controllers for modular multilevel converters', *IEEE Transactions on Energy Conversion*, vol. 36, no. 2, pp. 1562–1571, 2020.

[66] J. Heaton, *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.

[67] C. D. Doan and S.-y. Liong, 'Generalization for multilayer neural network bayesian regularization or early stopping', in *Proceedings of Asia Pacific association of hydrology and water resources 2nd conference*, 2004, pp. 5–8.

[68] K. Dumper, W. Jenkins, A. Lacombe, M. Lovett and M. Perimutter, *INTRODUCTORY PSYCHOLOGY*, 1st ed. Washington State University (WSU), 2019.

[69] R. S. Sutton, *Reinforcement Learning: Past, Present and Future*, B. McKay, X. Yao, C. S. Newton, J.-H. Kim and T. Furuhashi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 195–197.

[70] M. L. Minsky, *Theory of neural-analog reinforcement systems and its application to the brain-model problem*. Princeton University, 1954.

[71] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[72] P. J. Werbos, 'Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research', *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, no. 1, pp. 7–20, 1987.

[73] C. J. C. H. Watkins, 'Learning from delayed rewards', Ph.D. dissertation, King's College, 1989.

[74] L. Kaelbling, T. Lozano-Perez, I. Chuang and B. Duane, *Introduction to machine learning*, Lecture 10, MIT, Department of Electrical Engineering and Computer Science, 18th Dec. 2019. [Online]. Available: https://ocw.mit.edu/courses/6-036-introduction-to-machine-learning-fall-2020/ (visited on 20th Jan. 2022).

[75] J. Quintero Bermejo, 'Design of a flexible dc to ac converter and its control using reinforcement learning', M.S. thesis, Universitat Politècnica de Catalunya, 2020.

[76] C. Shyalika, T. Silva and A. Karunananda, 'Reinforcement learning in dynamic task scheduling: A review', *SN Computer Science*, vol. 1, no. 6, pp. 1–17, 2020.

[77] B. Jang, M. Kim, G. Harerimana and J. W. Kim, 'Q-learning algorithms: A comprehensive classification and applications', *IEEE Access*, vol. 7, pp. 133 653–133 667, 2019.

[78] A. Perera and P. Kamalaruban, 'Applications of reinforcement learning in energy systems', *Renewable and Sustainable Energy Reviews*, vol. 137, p. 110 618, 2021.

[79] R. Dearden, N. Friedman and S. Russell, 'Bayesian q-learning', in *Aaai/iaai*, 1998, pp. 761–768.

[80] H. Nguyen and H. La, 'Review of deep reinforcement learning for robot manipulation', in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, IEEE, 2019, pp. 590–595.

[81] C. G. Atkeson and J. C. Santamaria, 'A comparison of direct and model-based reinforcement learning', in *Proceedings of international conference on robotics and automation*, IEEE, vol. 4, 1997, pp. 3557–3564.

[82] Y. Lu, M. S. Squillante and C. W. Wu, 'A control-model-based approach for reinforcement learning', *arXiv preprint arXiv:1905.12009*, 2019.

[83] J. G. Vlachogiannis and N. D. Hatziargyriou, 'Reinforcement learning for reactive power control', *IEEE transactions on power systems*, vol. 19, no. 3, pp. 1317–1325, 2004.

[84] C. Wei, Z. Zhang, W. Qiao and L. Qu, 'An adaptive network-based reinforcement learning method for mppt control of pmsg wind energy conversion systems', *IEEE Transactions on Power Electronics*, vol. 31, no. 11, pp. 7837–7848, 2016.

[85] A. Kushwaha, M. Gopal and B. Singh, 'Q-learning based maximum power extraction for wind energy conversion system with variable wind speed', *IEEE Transactions on Energy Conversion*, vol. 35, no. 3, pp. 1160–1170, 2020.

[86] P. Kofinas, S. Doltsinis, A. Dounis and G. Vouros, 'A reinforcement learning approach for mppt control method of photovoltaic sources', *Renewable Energy*, vol. 108, pp. 461–473, 2017.

[87] A. Youssef, M. E. Telbany and A. Zekry, 'Reinforcement learning for online maximum power point tracking control', *J. Clean Energy Technol*, vol. 4, pp. 245–248, 2016.

[88] D. Alfred, D. Czarkowski and J. Teng, 'Model-free reinforcement-learning-based control methodology for power electronic converters', in *2021 IEEE Green Technologies Conference (GreenTech)*, IEEE, 2021, pp. 81–88.

[89] F. L. Lewis and D. Vrabie, 'Reinforcement learning and adaptive dynamic programming for feedback control', *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[90] G. Hübner, 'Adaptive dynamic programming', *Optimization and Operations Research*, vol. 4, p. 119, 2009.

[91] F.-Y. Wang, H. Zhang and D. Liu, 'Adaptive dynamic programming: An introduction', *IEEE computational intelligence magazine*, vol. 4, no. 2, pp. 39–47, 2009.

[92] M. Benosman, *Learning-based adaptive control: An extremum seeking approach–theory and applications*. Butterworth-Heinemann, 2016.

[93] P. Werbos, 'Approximate dynamic programming for realtime control and neural modelling', *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, pp. 493–525, 1992.

[94] S. Saadatmand, P. Shamsi and M. Ferdowsi, 'The heuristic dynamic programming approach in boost converters', in *2020 IEEE Texas Power and Energy Conference (TPEC)*, IEEE, 2020, pp. 1–6.

[95] S. Saadatmand, M. S. S. Nia, P. Shamsi, M. Ferdowsi and D. C. Wunsch, 'Heuristic dynamic programming for adaptive virtual synchronous generators', in *2019 North American Power Symposium (NAPS)*, IEEE, 2019, pp. 1–6.

[96] C. Lu, J. Si and X. Xie, 'Direct heuristic dynamic programming for damping oscillations in a large power system', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 1008–1013, 2008.

[97] M.Amin, *Control system for small and large converters*, ELK-21, Electronics for Control of Power, Lecture 4, NTNU, Department of Electric Power Engineering, 20th Sep. 2021.

[98] C. Bajracharya, M. Molinas, J. A. Suul, T. M. Undeland *et al.*, 'Understanding of tuning techniques of converter controllers for vsc-hvdc', in *Nordic Workshop on Power and Industrial Electronics (NORPIE/2008), June 9-11, 2008, Espoo, Finland*, Helsinki University of Technology, 2008.

[99] D. Atabay, *Pyrenn project*, https://github.com/yabata/pyrenn, Accessed: 2022–03-15, 2016.

[100] C. Bordons and C. Montero, 'Basic principles of mpc for power converters: Bridging the gap between theory and practice', *IEEE Industrial Electronics Magazine*, vol. 9, no. 3, pp. 31–43, 2015.

[101] J. Holtz, 'A predictive controller for the stator current vector of ac machines fed from a switched voltage source', *Proc. of IEE of Japan IPEC-Tokyo'83*, pp. 1665–1675, 1983.

[102] F. Toso, A. Favato, R. Torchio, P. Alotto and S. Bolognani, 'Continuous control set model predictive current control of a microgrid-connected pwm inverter', *IEEE Transactions on Power Systems*, vol. 36, no. 1, pp. 415–425, 2020.

[103] P. Cortés, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo and J. Rodríguez, 'Predictive control in power electronics and drives', *IEEE Transactions on industrial electronics*, vol. 55, no. 12, pp. 4312–4324, 2008.

[104] T. Orlowska-Kowalska, F. Blaabjerg and J. Rodríguez, *Advanced and intelligent control in power electronics and drives*. Springer, 2014, vol. 531.

[105] B. Long, Z. Zhu, W. Yang, K. T. Chong, J. Rodríguez and J. M. Guerrero, 'Gradient descent optimization based parameter identification for fcs-mpc control of lcl-type grid connected converter', *IEEE Transactions on Industrial Electronics*, vol. 69, no. 3, pp. 2631–2643, 2021.

[106] A. Mora, R. Cárdenas-Dobson, R. P. Aguilera, A. Angulo, F. Donoso and J. Rodriguez, 'Computationally efficient cascaded optimal switching sequence mpc for grid-connected three-level npc converters', *IEEE Transactions on Power Electronics*, vol. 34, no. 12, pp. 12 464–12 475, 2019.

[107] L. Maccari Jr, D. Lima, G. Koch and V. Montagner, 'Robust model predictive controller applied to three-phase grid-connected lcl filters', *Journal of Control, Automation and Electrical Systems*, vol. 31, no. 2, pp. 447–460, 2020.

[108] C. Xue, L. Ding and Y. Li, 'Ccs-mpc with long predictive horizon for grid-connected current source converter', in *2020 IEEE Energy Conversion Congress and Exposition (ECCE)*, IEEE, 2020, pp. 4988–4993.

[109] M. M. Seron, *Receding horizon control*, https://www-eng.newcastle.edu.au/eecs/cdsc/books/cce/Slides/RecedingHorizonControl.pdf, Accessed: 2022–04-27, 2004.

[110] B. Foss and T. A. N. Heirung, 'Merging optimization and control', *Lecture Notes*, 2013.

[111] H. A. Young, M. A. Perez, J. Rodriguez and H. Abu-Rub, 'Assessing finite-control-set model predictive control: A comparison with a linear current controller in two-level voltage source inverters', *IEEE Industrial Electronics Magazine*, vol. 8, no. 1, pp. 44–52, 2014.

[112] S. K. Gannamraju, D. Valluri and R. Bhimasingu, 'Comparison of fixed switching frequency based optimal switching vector mpc algorithms applied to voltage source inverter for standalone applications', in *2019 National Power Electronics Conference (NPEC)*, IEEE, 2019, pp. 1–6.

[113] J. Rodriguez, R. Heydari, Z. Rafiee, H. A. Young, F. Flores-Bahamonde and M. Shahparasti, 'Model-free predictive current control of a voltage source inverter', *IEEE Access*, vol. 8, pp. 211 104–211 114, 2020.

[114] K. S. Alam, D. Xiao, M. Parvez Akter, D. Zhang, J. Fletcher and M. F. Rahman, 'Modified mpc with extended vvs for grid-connected rectifier', *IET Power Electronics*, vol. 11, no. 12, pp. 1926–1936, 2018.

[115] M. Nauman and A. Hasan, 'Efficient implicit model-predictive control of a three-phase inverter with an output lc filter', *IEEE Transactions on Power Electronics*, vol. 31, no. 9, pp. 6075–6078, 2016.

[116] M. Fairbank, S. Li, X. Fu, E. Alonso and D. Wunsch, 'An adaptive recurrent neural-network controller using a stabilization matrix and predictive inputs to solve a tracking problem under disturbances', *Neural Networks*, vol. 49, pp. 74–86, 2014.

[117] S. Li, M. Fairbank, C. Johnson, D. C. Wunsch, E. Alonso and J. L. Proao, 'Artificial neural networks for control of a grid-connected rectifier/inverter under disturbance, dynamic and power converter switching conditions', *IEEE transactions on neural networks and learning systems*, vol. 25, no. 4, pp. 738–750, 2013.

[118] R. Pascanu, T. Mikolov and Y. Bengio, 'On the difficulty of training recurrent neural networks', in *International conference on machine learning*, PMLR, 2013, pp. 1310–1318.

[119] S. Hochreiter and J. Schmidhuber, 'Long short-term memory', *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[120] B. Hanin, 'Which neural net architectures give rise to exploding and vanishing gradients?', *Advances in neural information processing systems*, vol. 31, 2018.

[121] G. Hübner, 'Adaptive dynamic programming', *OPTIMIZATION AND OPERATIONS RESEARCH–Volume IV*, p. 119, 2009.

[122] J. J. Govindhasamy, S. F. McLoone and G. W. Irwin, 'Second-order training of adaptive critics for online process control', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 2, pp. 381–385, 2005.

[123] The MathWorks, Inc, *Simulink user's guide (r2022a)*, The MathWorks, Inc, Natick, Massachusets, 2022.

Ella Gardner Øxnevad

Artificial Neural Network-Based Control for Grid-Connected Converter

NTNU
Norwegian University of
Science and Technology