The Energy Sponsor Programme

AALBORG UNIVERSITY
STUDENT MANUSCRIPT

Neural Network-Based Imitation of Model Predictive Control for Power Converters

A. Teran[a,*], C. Roche[a], P. Izquierdo[a]

[a] *Aalborg University, Faculty of Engineering and Science, Niels Jernes Vej 12, DK-9220 Aalborg East, Denmark*

Abstract

Machine learning and neural networks have been at the forefront of many technological breakthroughs in recent years. This study aims to make use of these techniques to implement the control scheme of power converters, imitating the more widespread model predictive control method. Model predictive control (MPC) has been shown to offer great advantages compared to more traditional techniques, but it has an important drawback in its high computational requirements. By predicting the behavior of the system for longer periods of time, performance can be improved at the cost of an exponentially increasing computational time. This can be potentially overcome by imitating such a method using a neural network, which can greatly reduce computational cost. Training data is collected using an accurate simulation model of conventional MPC schemes, for a wide range of parameters. This data is used to train an artificial neural network using the Python implementation of Tensorflow, which allows for a highly flexible neural network structure. The trained neural network is then experimentally verified by implementing it in a laboratory environment.

Keywords: Finite-set model predictive control, artificial neural networks, inverter control, imitation learning

1. Introduction

Finite-set model predictive control (FS-MPC) is an advanced control method that has recently been gaining attention in the field of power electronics. It offers advantages over more traditional methods due to its straightforward design, easy inclusion of control objectives, and discrete nature, which makes it a natural fit for power converters. Moreover, FS-MPC allows for simple non-linear and multi-variable closed-loop control design [1].

The main shortcoming of FS-MPC are its high computational requirements, especially for the control of multilevel power converters and/or multi-step prediction. This paper presents one way to overcome this issue by designing an artificial neural network-based imitator of FS-MPC, that is able to retain the performance of the algorithm while reducing its computational burden.

ANNs are universal function approximators, as proven formally in 1993 by Leshno et al. [2]. The parallel structure of artificial neural networks allows for high-speed processing. This means that neural networks are computationally very cheap, allowing for complex structures to be executed in short processing times.

Applications of neural networks to control systems have been considered for years, especially with regards to modelling of nonlinear systems, system identification, and design of control structures. Some areas of power electronics where neural networks have proven to be useful are: delayless filtering and waveform processing, design of estimators for motor drives (such as position, speed, or flux), as well as controller design for power converters [3].

The potential of neural networks in the control of power converters has been shown in the following publications, to select a few: [4] details the development of a recurrent neural network for control of a grid-connected DC-to-AC converter without making use of the back-propagation algorithm, [5] uses a neural network to control a DC-to-DC converter for use in photovoltaic applications, while [6] uses a neural network model of a photovoltaic system to obtain optimal control actions. Most similarly to this project, [7] shows how a neural network imitator of FS-MPC can be used to control a multilevel converter, albeit with a relatively low imitation accuracy of around 90%.

This paper proposes an ANN-based imitator of FS-MPC for the control of a two-level, six-switch, DC-to-AC power converter. The neural network is trained offline using data obtained from FS-MPC algorithms with a prediction horizon of 1, 2, and 3 steps ahead.

This paper is structured as follows: section 2 describes the converter modeling, section 3 introduces the operation principles of the FS-MPC, section 4 shows the design of the ANN imitator, and section 5 presents the experimental results, with section 6 outlining the research conclusions.

The control algorithms are implemented as *Matlab* code. The ANN-based imitators are designed and trained using the implementation of the *TensorFlow* library for *Python 3*, and making extensive use of the *Keras* API. All of the controllers are verified using *Simulink* and *Simscape* and then tested on an experimental setup based on *dSpace*.

☆This manuscript is submitted for the CES student conference organised by Department of Energy Technology, Aalborg University.

*Corresponding author:

Email addresses: ateran18@student.aau.dk (A. Teran), croche18@student.aau.dk (C. Roche), pizqui18@student.aau.dk (P. Izquierdo)

2. Inverter description and modelling

The converter to be controlled is a DC-to-AC two-level three-phase inverter. Its main goal is to generate a three-phase sinusoidal waveform that emulates the reference voltage signal with a frequency and amplitude given.

The 8 possible combinations of the switches are called converter states. For the converter to synthesize sinusoidal output voltage and current signals, it must rapidly switch between its 8 states. This results in discrete-valued signals with high-frequency harmonics that must be filtered out in order to obtained the desired sine waves.

The second order low-pass filter used for this purpose is accomplished through an LC-circuit. This filter structure is connected to each one of the converter phases output terminals in order to filter out the high-frequency harmonics of the three phases.

It consists of an inductor with inductance $L_f = 2.4$ mH and parasitic resistance $R_f = 0.1\ \Omega$ in series with the inverter; and a capacitor in parallel with the load, with capacitance $C_f = 14.2\ \mu$F. The output phase current from the inverter i_L passes through the inductor, and creates a voltage drop across it v_L. The voltage drop across the capacitor is the same as that of the load, v_C. The voltage drop across the filter v_i (inverter output voltage) is the addition of v_C and v_L. The current that flows to the load R_{load} is called i_{load}.

Using this notation, the LC filter can be mathematically described as:

$$\begin{cases} L_f \frac{di_L}{dt} = -R_f i_L - v_C + v_i \\ C_f \frac{dv_C}{dt} = i_L - i_{load} \end{cases} \tag{1}$$

Using the state-space convention, these equations can be rewritten to the following form:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t) \\ \mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{D} \cdot \mathbf{u}(t) \end{cases} \tag{2}$$

Where the vector of system states $\mathbf{x}(t)$, the vector of system inputs $\mathbf{u}(t)$, and the output $\mathbf{y}(t)$ are defined as:

$$\mathbf{x}(t) = \begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} \quad \mathbf{u}(t) = \begin{bmatrix} v_i(t) \\ i_{load}(t) \end{bmatrix} \tag{3}$$

$$y(t) = v_{load}(t) \tag{4}$$

The coefficient matrices of the state-space model are then:

$$\mathbf{A} = \begin{bmatrix} -\frac{R_f}{L_f} & -\frac{1}{L_f} \\ \frac{1}{C_f} & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \frac{1}{L_f} & 0 \\ 0 & -\frac{1}{C_f} \end{bmatrix} \tag{5}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 & 0 \end{bmatrix} \tag{6}$$

This state-space model is then discretized to:

$$\mathbf{x}((k+1)T) = \mathbf{G}(T) \cdot \mathbf{x}(kT) + \mathbf{H}(T) \cdot \mathbf{u}(kT) \tag{7}$$

Where the discretized coefficient matrices $\mathbf{G}(T)$ and $\mathbf{H}(T)$ become functions of the sampling time, and can be found to be:

$$\mathbf{G}(T) = e^{\mathbf{A}T}, \quad \mathbf{H}(T) = \mathbf{A}^{-1}\left(e^{\mathbf{A}T} - \mathbf{I}\right)\mathbf{B} \tag{8}$$

Equation 7, with the coefficients shown in Equation 8, is used in the FS-MPC algorithm to predict the future values of the states by making use of their current values and possible inputs.

3. FS-MPC operation principles

FS-MPC makes use of the state-space model of the converter to predict the future values of the system states for every possible control action. It then calculates the cost function value that reflects control error. As such, FS-MPC can be understood as an algorithm that solves the online optimization problem defined by the chosen cost function. For the two-level, six-switch, DC-to-AC converter, the possible control actions are each of the eight feasible converter output vectors. Since two of them output null voltage, when all three top switches are either on or off, the number of control outputs can be reduced to seven.

The predictions of the system states are then compared to their corresponding reference values, with the algorithm selecting the switching state that minimizes this difference. If this control error is considered for a single step ahead of the current sampling time, the algorithm is said to have a one-step prediction horizon.

When making predictions for more than a single step ahead, the number of calculations to be performed online increases exponentially. This is due to the fact that each of the seven possible control actions in the first step branches off into seven more for the second step, resulting in a total of $7^2 = 49$ evaluations of the model. This pattern repeats for further steps. Therefore, the total number of model evaluations follows an exponential law of the form n^h, where n is the number of possible states of the system and h is the prediction horizon.

The consequences of this exponential computational cost are clear: FS-MPC for large prediction horizons and/or multi-level converters can be very challenging to implement for real-time control within the short sampling times required to obtain adequate harmonic performance.

A diagram of the calculation steps that one-step FS-MPC follows for each time step is shown in Figure 1.

The set of input variables that are measured are named $\mathbf{x}_i(t_{k+h})$, where i are the switching states for which the variables are calculated, h is the time step, and k is the current sampling instant. $\hat{\mathbf{x}}_i(t_{k+h})$ uses the same nomenclature but refers to the predicted input variables calculated using the discrete state-space model.

The basic operation of the FS-MPC algorithm starts at the beginning of every sampling period by receiving

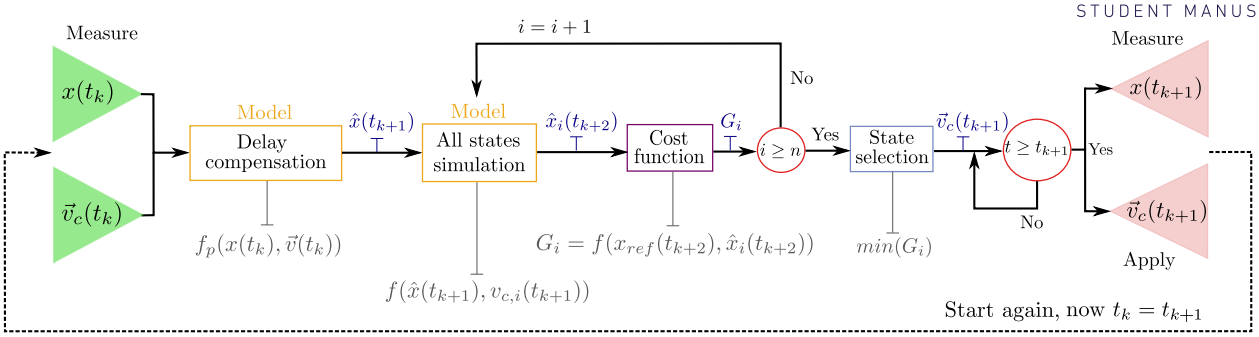

Figure 1: Information flow along one step time calculation for FS-MPC with one step horizon prediction. Figure based on [8].

the new measurements (at time t_k): the three-phase inductor currents ($i_{L,a}, i_{L,b}$, $i_{L,c}$), and the capacitor voltages ($v_{C,a}, v_{C,b}$, $v_{C,c}$), which are the previous values of the states in the state-space model. The algorithm also requires measurements of the load currents ($i_{load,a}, i_{load,b}$, $i_{load,c}$) and to know the previously applied output voltage of the converter ($v_{i,a}, v_{i,b}$, $v_{i,c}$). The output voltage of the converter defined by the switching state and the load currents form the input vector of the state-space model.

Since the system is balanced, and therefore its zero component is null, the three phase variables of each component can be transformed to the $\alpha\beta$ reference frame. By making use of this transformation, each vector that was previously formed by three components a, b, c, now only has two components α, β. Therefore, the total number of inputs and states of the system is reduced from 12 to 8: $\vec{i}_{L,\alpha\beta}$, $\vec{v}_{C,\alpha\beta}$, $\vec{i}_{load,\alpha\beta}$, and $\vec{v}_{i,\alpha\beta}$.

Once the algorithm receives all the measurements from the previous sampling time in $\alpha\beta$, by means of the state-space model predicts the values of the states in the next sampling time. This is needed because in a digital system it is unfeasible to read measurements, perform calculations and apply commands at the same sampling instance, due to the time needed for computation ~~and because signals can only be updated at the beginning of a sampling period~~. The model prediction of $\mathbf{x}(t_{k+1})$ is called delay compensation and gives the predicted value of those variables [9], named $\hat{\mathbf{x}}(t_{k+1})$.

Once the predicted values for $t(k+1)$ are obtained the state-space model is applied again in an iterative manner, in order to obtain the values of the states at $t(k+2)$ for each control action, i.e. the applied voltage state i. This process is illustrated in Figure 2.

For a multi-step FS-MPC algorithm this process would be repeated to obtain the values at future sampling times taking the previous predictions as states.

3.1. Cost function

The basic control objective of the algorithm is to select the proper switching state that generates a voltage vector ($\vec{v}_C(t_{k+2})$) such that the phase voltage tracks the reference voltage with minimal error. The reference voltage signals are given at t_k for $\vec{v}_{ref}(t_{k+2})$ in the $\alpha\beta$ reference frame.

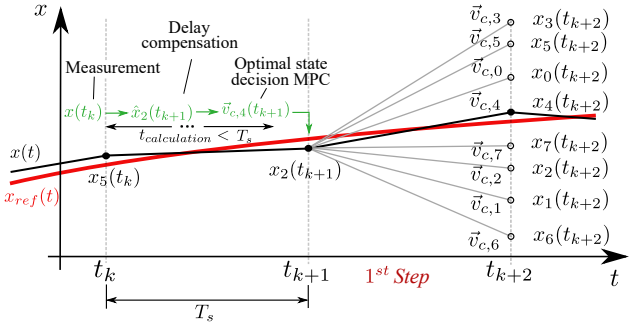

Figure 2: Time based representation of the FS-MPC algorithm performance. Figure based on [8].

The predicted values are evaluated through the use of a cost function and the voltage vector state that produces minimum error with the reference is selected and applied in t_{k+1}. As an example, in Figure 2 the voltage vector with minimum cost would be $\vec{v}_{C,4}$.

The cost function can penalize any terms as desired, and should be constructed in a way that results in the FS-MPC outputs that best fit the requirements of the application for which the algorithm is to be designed.

The most basic cost function would be formed by the euclidean distance between the phase voltage and the reference one. In addition to this term, the derivative term of the load voltage is included to improve the performance of the algorithm. This is again penalized by minimizing the euclidean distance between the derivative of the voltage reference and the derivative of the load voltage. The term is added to the cost function with a weighting factor λ [9]. This cost function G is shown in Equation 9.

$$G = (v_{ref,\alpha} - v_{c,\alpha})^2 + (v_{ref,\beta} - v_{c,\beta})^2 + \lambda G_d \quad (9)$$

The expression for derivative term G_d is shown in Equation 10.

$$G_d = (C_f \omega_{ref} v_{ref,\beta} - i_{f\alpha} + i_{load,\alpha})^2 + (C_f \omega_{ref} v_{ref,\alpha} + i_{f\beta} - i_{load,\beta})^2 \quad (10)$$

Penalizing this derivative term results in ~~very~~ improved performance of the FS-MPC for a one-step horizon. For a single step, the selected switching state could be the

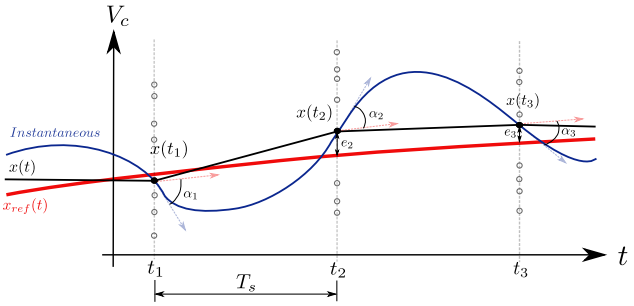

Figure 3: Effect of large voltage derivatives. Where α_i is difference between the reference voltage derivative and the load voltage derivative. e_i, error at the sampling instants. Picture based on [9].

best one for the current sampling instant but have a large derivative at that point, and therefore lead to worse values in the next sampling instant. This phenomenon is illustrated in Figure 3.

The addition of the derivative term to the cost function does not modify the behavior of multiple-step FS-MPC nearly as much as it does for the single-step algorithm. This is due to the fact that undesirable voltage derivatives are already penalized by considering the behavior of the converter in future steps. In the rare case where resonance could occur in-between sampling periods, the derivative term would still mitigate such behavior in multi-step FS-MPC.

The selection of the weighting factor between the two terms of the cost function is always a trade-off decision. In order to select the weighting factor that provides the better performance, measured using the lowest THD value in the phase voltages, a test with different weighting factors in the FS-MPC model is performed in simulation, which resulted in $\lambda = 1$ as the optimum value.

3.2. Advantages and drawbacks

The main advantages of FS-MPC are its robustness, excellent transient characteristics, and the possibilities it offers to account for non-linearities and constraints [9]. By allowing its cost function to be designed as desired, the algorithm is extremely flexible and can include arbitrarily many control objectives. It can also greatly simplify the control of multivariable systems, as FS-MPC can be designed to output as many values as required.

On the other hand, its main disadvantage is the large computational burden it places on the controller hardware. This also limits the sampling frequencies at which the converter can operate. Since FS-MPC outputs the control actions directly and does not make use of any modulation techniques, the smaller the sampling time of the system, the better its performance. This is the main ~~trade-off~~ that limits the broader implementation of FS-MPC algorithms in power electronics. Another disadvantage of FS-MPC is that the algorithm heavily relies on the accuracy of the model it uses to make predictions. If the model is inaccurate, the algorithm will result in poor performance.

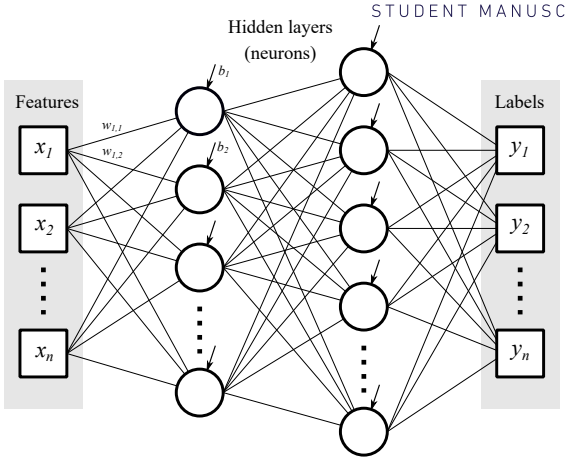

Figure 4: Artificial neural network diagram.

One approach to minimize the drawbacks of FS-MPC introduced by its large computational requirements is to solve the optimization problem offline [8]. Such an approach is taken in this project, using data obtained by offline simulations of the FS-MPC described in this chapter to train an artificial neural network-based imitator controller.

4. Imitation learning method

This section covers the concept of imitation learning and the design of the ANN-based imitator controllers proposed with the goal of overcoming the computational drawbacks of FS-MPC.

4.1. Artificial neural networks

Artificial neural networks (ANN) are machine learning-based computing structures that are loosely inspired on the biological neural networks found in animal brains. They are formed by a series of nodes called units or artificial neurons that are arranged in different layers. In a fully-connected ANN, the values of units in each layer depend on the values of all units in the previous layer. Each layer can then be defined by three parameters: its weight matrix $\mathbf{W}$, bias vector $\mathbf{b}$ and activation function f. The values of units $\mathbf{a}$ in a layer are simply computed as:

$$\mathbf{a} = f(\mathbf{z}) = f(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}) \tag{11}$$

Where $\mathbf{x}$ is a vector containing the values of all units in the previous layer. Each neuron can be interpreted as an operation scaling the value of each unit in its previous layer by a single scalar weight, adding all of these weighted values together, and adding to them a scalar bias. The obtained value is then used as the argument of the activation function. For the ANN to be able to represent non-linear relationships, this activation function must also be non-linear. More formally, a neural network with a sufficiently large

number of neurons and containing non-polynomial activation functions can approximate any arbitrarily complex relationship in a data set, as proven in [2].

Sigmoid and hyperbolic tangent activation functions are widely used. However, both functions may present the so called vanishing gradient problem since they strongly saturate at values of $\mathbf{z}$ that deviate from zero. This can make gradient-based learning, the most commonly used method for ANN training, very difficult. Instead, the rectified linear unit (ReLU) is used since it minimizes this issue and is computationally lighter. The ReLU function is defined in Equation 12.

$$ReLU(x) = \begin{cases} 0 & \forall x \leq 0 \\ x & \forall x > 0 \end{cases} \tag{12}$$

4.1.1. The cost function

The process of offline training is based on data points referred to as training examples. Input data is often referred to as *features* and the desired outputs for each example are called *labels*. The outputs obtained at the last layer of network are referred to as predictions.

Training a neural network means adjusting its weight and bias parameters in such a way that minimizes the deviation between predictions and labels. This deviation is penalized by means of a cost function J. This optimization problem is often non-convex, which implies that it cannot be solved analytically [10]. Therefore, the choice of cost function heavily influences the convergence of the employed optimization method.

The problem at hand is multi-class classification, where each training example can only belong to a single category. In more concrete terms, each set of input parameters of the FS-MPC algorithm can only result in a single converter state command. The cost function chosen is then categorical cross-entropy, which takes this fact into consideration. Categorical-cross entropy (CCE) is defined as:

$$J = CCE = -\log\left(\frac{e^{s_p}}{\sum_{j=1}^{n} e^{s_j}}\right) \tag{13}$$

Where $\mathbf{s}$ are the predictions of the network, s_p is the prediction corresponding to the positive class of the label vector, and n is the number of predictions or elements of $\mathbf{s}$.

4.1.2. Back-propagation

Back-propagation [11] is an algorithm that allows the cost function information to flow backwards from the outputs. Its goal is to obtain the gradient of the cost function J with respect to the weight matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$. This gradients are then used to update the values of the parameters, by making use of an optimization algorithm.

Let w_{ij} denote the weight attached to the connection between the i-th node in layer $L-1$ and the j-th node in layer L. The gradient of the cost function with respect to this weight and the cost with respect to the bias b_j can be found using the chain rule of calculus:

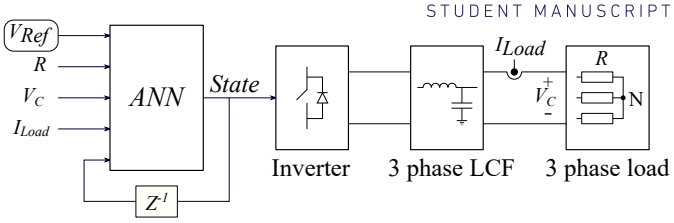

Figure 5: Features sources of the ANN.

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial a_j}{\partial z_j}\frac{\partial J}{\partial a_j} \qquad \frac{\partial J}{\partial b_j} = \frac{\partial z_j}{\partial b_j}\frac{\partial a_j}{\partial z_j}\frac{\partial J}{\partial a_j} \tag{14}$$

All of the right-hand side terms of Equation 14 can be computed explicitly once the cost and activation functions are defined. For a layer $L-1$, where layer L is the layer immediately closer to the output, the gradient of the cost function with respect to the value of each unit can be found to be:

$$\frac{\partial J}{\partial a_j} = \sum_{l \in L}\left(\frac{\partial J}{\partial a_l}\frac{\partial a_l}{\partial z_l}\frac{\partial z_l}{\partial a_j}\right) = \sum_{l \in L}\left(\frac{\partial J}{\partial a_l}\frac{\partial a_l}{\partial z_l}w_{jl}\right) \tag{15}$$

Therefore, back-propagation can only obtain the gradients of each layer once those in the layer immediately closer to the output are known. This is where the algorithm gets its name from, as cost function gradients propagate backwards from the last to the first layers.

Using a steepest descent algorithm, the weights and biases would then updated for each algorithm iteration k as:

$$\begin{aligned} \boldsymbol{W}^{k+1} &= \boldsymbol{W}^{k} - \eta\frac{\partial \boldsymbol{J}^{k}}{\partial \boldsymbol{W}^{k}} \\ \boldsymbol{b}^{k+1} &= \boldsymbol{b}^{k} - \eta\frac{\partial \boldsymbol{J}^{k}}{\partial \boldsymbol{b}^{k}} \end{aligned} \tag{16}$$

Where η is a scalar constant often referred to as learning rate. The ANNs presented in this project are trained using the *Adam* algorithm [12], that aims to improve on steepest descent. This algorithm tunes the learning rate in an adaptive manner that prevents the parameters of the network from converging at globally poor minima of the cost function.

4.2. The ANN-based imitator

Based on these principles, this section details the training of the ANN-based imitator controllers.

4.2.1. Training data acquisition

ANN data harvesting was carried out using previously experimentally validated simulations of the FS-MPC algorithms. In addition to avoiding putting the hardware at risk, training with simulated data allows to obtain as many points as desired, as well as to get them evenly distributed through the feature data space. Moreover, using this procedure, data can be obtained from controllers that cannot be executed in real time due to high computational burden. This data can then be used to train an ANN.

Similarly to the FS-MPC controller inputs, the features of the data points consist on load resistance R, reference voltages $\vec{v}_{ref,\alpha\beta}$, measured voltages $\vec{v}_{C,\alpha\beta}$ and measured current at the load $\vec{i}_{load,\alpha\beta}$. Furthermore, the previous selected state is fed back as another feature. The ANN data flow is illustrated in Figure 5.

The obtained training data set consists of an evenly distributed 8-dimensional matrix formed by permutations of the input features. The network was trained for a fixed 325 V amplitude, 50 Hz sine wave as $\vec{v}_{ref,\alpha\beta}$, $R \in [30, 60]$ Ω, voltage control error $(\vec{v}_{ref,\alpha\beta} - \vec{v}_{C,\alpha\beta}) \in [-5, 5]$ V, and $i_{load,\alpha\beta} \in [-16, 16]$ A. These limits were set by the used experimental setup. The training data set contains several million data points, making over-fitting training highly improbable.

The validation and test data sets were each obtained as half a million random points within the limits of each variable, which differ from the values set by the training grid.

4.2.2. Optimizing the network structure

Computational cost and performance were the considered parameters to find the optimal network layout. The main metric used to define imitation performance is accuracy. Accuracy is defined for a set of examples as the ratio between correct predictions and total number of predictions. The optimization of the ANN structure was performed using data obtained from the 1 step ahead FS-MPC model. The addition of more than a single hidden layer does not significantly increase the accuracy of the model, but still raises the computational burden of the network. Hence, a configuration with a single hidden layer was selected for implementation.

Training error, test error and total harmonic distortion (THD) were taken into account to find the optimal number of neurons in the hidden layer, while considering that a lower number of neurons also entails lower computational load. Error is defined as 1 - accuracy. Figure 6 displays the performance of the network with a changing number of neurons in the middle layer. From this data, 15 neurons are considered to be the optimum trade-off.

The obtained neural-network imitators have an accuracy of 97.96% for 1-step FS-MPC, 97.10% for 2-step, and 97.57% for 3-step.

Deeper understanding of the ANN model accuracy is acquired through the confusion matrix. Facing each true label against its corresponding network prediction illustrates the way in which the system fails to evaluate new data points. In this way, Figure 7 shows how the wrongly predicted labels of the trained model most often fall in neighbor states. The state 1, representing vector (0,0,0), has the lowest accuracy since it is under-trained compared to the other states. This is because the system is trained with a peak-to-peak voltage $V_{pp} = 650\ V$ while the inverter supply consists on a $V_{DC} = 700\ V$, thus resulting in only 0.62% of data points with state 1 non-zero labels.

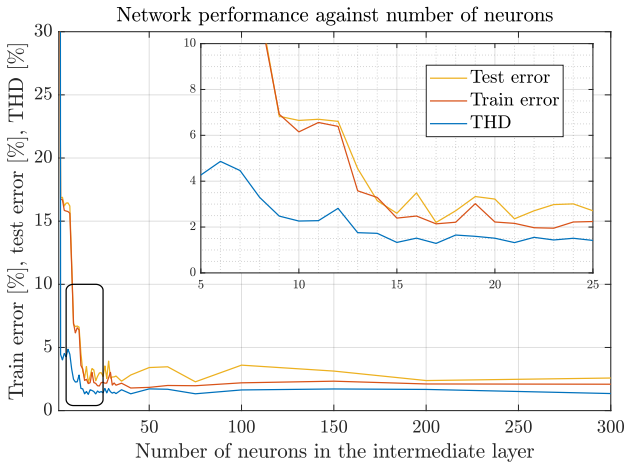

Figure 6: Training error, test error and average simulation THD against number of neurons in the hidden layer for a 1 step prediction ANN imitator.

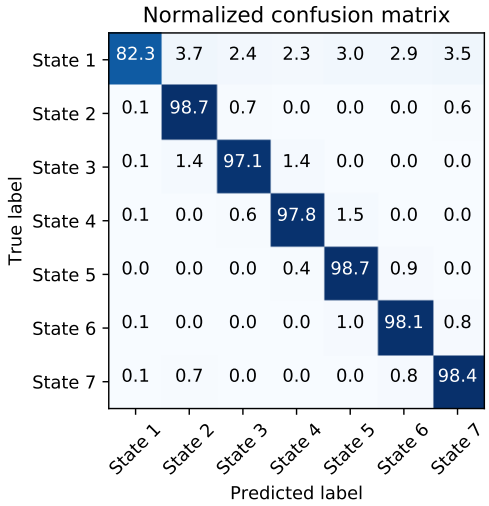

Figure 7: Test data confusion matrix of 1-step prediction ANN imitator.

5. Results

For validation of the model as well as performance comparison between the ANN imitators and FS-MPC controllers, *dSpace* is used for processing and implementing the model in the experimental setup. The algorithm sampling time is set to $T_s = 20\ \mu s$.

Experimental results were obtained from a four-channel oscilloscope displaying the three line-to-line voltages and a phase current. On top of this, the *dSpace* software measures data such as switching frequency, execution time and V_{DC}.

5.1. Steady state performance

Figures 8 and 9 show steady-state performance for 1-step FS-MPC and its imitator, respectively. It is observed

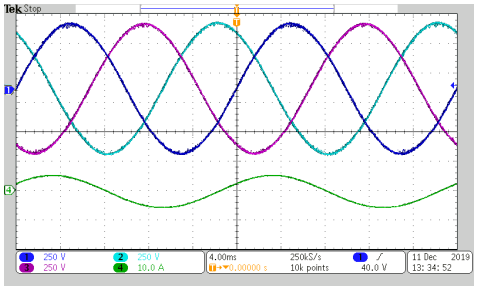

Figure 8: MPC for 1 step prediction steady-state performance with a 325 V reference amplitude and $R_{load} = 60\ \Omega$.

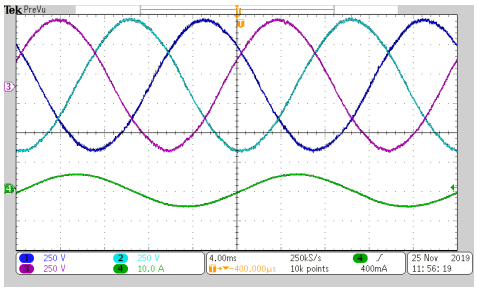

Figure 9: ANN imitator for 1 step prediction steady state performance with a 325 V reference amplitude and $R_{load} = 60\ \Omega$.

Steps	Controller	THD [%]	$f_{sw}[Hz]$	t_{ex} $[\mu s]$
1	MPC	1.654	8150	10
	ANN	1.356	8200	11
2	MPC	1.662	8100	13
	ANN	1.547	8400	11
3	MPC	-	-	>20
	ANN	1.84	8300	11

Table 1: Comparison between different steps ahead prediction in steady-state for FS-MPC and ANN imitation controllers.

that both controllers provide a similarly accurate performance, which confirms a successful learning process of the imitator. This close correlation between FS-MPC and its ANN imitator was expected from the high accuracy values obtained in the training process.

Similar results were obtained for FS-MPC and imitators for several steps. To illustrate their differences, the metrics chosen are THD and switching frequency f_{sw}, as listed in Table 1.

THD and switching frequency show similar performance for both methods and for different prediction horizons, with a slight increase in both metrics as prediction horizon increases. The ANN imitators have equal execution time regardless of prediction horizon, as the neural network maintains its structure. FS-MPC, however, has an execution time that increases exponentially with prediction horizon. For a three-step horizon, the algorithm cannot be evaluated in real time because its execution time is longer than the sampling time.

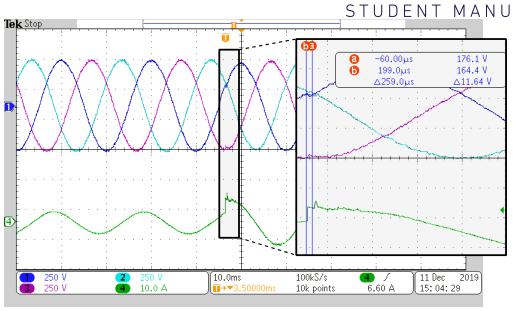

Figure 10: Load step transient response for the 1-step imitator.

5.2. Load step transient responses

One of the main advantages of FS-MPC over other control techniques is its ability to react quickly to unexpected disturbances or reference changes. Figure 10 shows the transient response of the 1-step imitator for a load step going from 60 Ω to 30 Ω.

A fast voltage transient response can be observed. Regarding current response, there is a spike after the transient which is considered a purely hardware-related issue due to the noise introduced by the load switching device.

The comparison between transient responses between FS-MPC and ANN imitators shows virtually equal performance. Increasing the prediction horizon considerably improves transient response for both methods. Consequently, voltage settles faster and the current spike is appreciably lower.

In order to obtain more detailed information about transient characteristics of the step response, the signals must be converted into dq-frame. Nevertheless, since the transients have a very short time span, the noise of the signals impeded the visualization of responses when dq-frame voltages and currents were measured using the *dSpace* tool.

For the considered cases, transient performance improvements obtained by increasing the prediction horizon come at the cost of lower steady-state THD.

5.3. Robustness

The ability to adjust to unexpected characteristics is fundamental for a controller. Experimental results were obtained by varying the reference signal and observing the controller performance. Fast and accurate tracking is observed when introducing a frequency step, as shown in Figure 11. Similar outcomes were obtained for steady-state performance. Table 2 compares the THD of FS-MPC and its imitators at different frequencies, resulting in the ANN consistently outperforming the MPC.

Comparable results were obtained for variations in reference voltage. Table 3 displays the resultant THD. Tests for voltages higher than 325 V were not carried out due to hardware limitations, but similar results are to be expected.

It is important to point out that the satisfactory response of the FS-MPC robustness-wise was expected, since it has reference voltage amplitude and frequency as inputs.

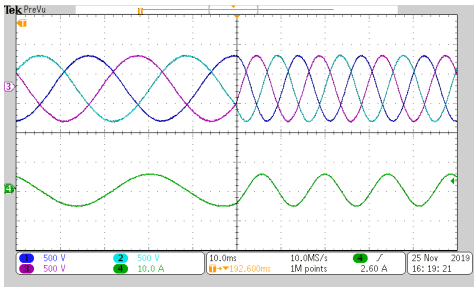

Figure 11: Frequency step from 30 to 70 Hz for the 1-step imitator.

f_{ref} $[Hz]$	Controller	THD [%]	$f_{sw}[Hz]$
20	MPC	1.679	8100
	ANN	1.418	8150
40	MPC	1.676	8100
	ANN	1.524	8200
50	MPC	1.640	8150
	ANN	1.356	8200
60	MPC	1.607	8100
	ANN	1.355	8300
80	MPC	1.523	8300
	ANN	1.447	8400

Table 2: Steady-state performance of MPC and ANN models for different values of reference frequency f_{ref}.

$V_{ref}[V]$	THD [%]	$f_{sw}[Hz]$
100	3.124	13400
200	2.007	12500
300	1.564	9200

Table 3: Steady-state performance of the 1-step imitator for different reference voltages.

It is however remarkable to find such good results for the ANN controller. Thereby, confirming the mapping ability of the network, since it has correctly extrapolated for untrained inputs.

6. Conclusion

This report documents the implementation of an FS-MPC algorithm for a 2-level DC-to-AC power converter, with the goal of designing ANN-based imitator controllers with comparable performance. As demonstrated experimentally, the obtained imitators closely match the behavior of the original FS-MPC.

The ANN imitators are shown to be robust to changes in voltage reference amplitude and frequency, validating their ability to perform adequately under conditions for which they are not explicitly trained. This illustrates the capacity of ANNs to learn non-linear mappings and extrapolate outside of their training data.

Moreover, this study shows that a 3-step FS-MPC algorithm with an unfeasible real-time implementation, due to computational limitations, may be successfully imitated by a feasible neural network-based controller trained using adequate data. This same method can be used in a similar manner to obtain imitators of more complex controllers, thus reducing the weight of limitations imposed by computational requirements.

References

[1] Jose Rodriguez and Patricio Cortes Estay. *Predictive control of power converters and electrical drives*. IEEE/Wiley, 2012. ISBN 9781119941446.

[2] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function. Technical report, 1993.

[3] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat Abd Elatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey, nov 2018. ISSN 24058440.

[4] Xingang Fu, Shuhui Li, Michael Fairbank, Donald C. Wunsch, and Eduardo 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*, 26(9):1900–1912, sep 2015. ISSN 21622388. doi: 10.1109/TNNLS.2014.2361267.

[5] Neeraj Priyadarshi, Sanjeevikumar Padmanaban, Jens Bo Holm-Nielsen, Vigna K. Ramachandaramurthy, and Mahajan Sagar Bhaskar. An AN-GA controlled SEPIC converter for photovoltaic grid integration. In *Proceedings - 2019 IEEE 13th International Conference on Compatibility, Power Electronics and Power Engineering, CPE-POWERENG 2019*. Institute of Electrical and Electronics Engineers Inc., apr 2019. ISBN 9781728132020. doi: 10.1109/CPE.2019.8862395.

[6] Yang Sun, Shuhui Li, Bo Lin, Xingang Fu, Malek Ramezani, and Ishan Jaithwa. Artificial Neural Network for Control and Grid Integration of Residential Solar Photovoltaic Systems. *IEEE Transactions on Sustainable Energy*, 8(4):1484–1495, oct 2017. ISSN 19493029. doi: 10.1109/TSTE.2017.2691669.

[7] Daming Wang, Xin Yin, Sai Tang, Chao Zhang, Z. John Shen, Jun Wang, and Zhikang Shuai. A Deep Neural Network Based Predictive Control Strategy for High Frequency Multilevel Converters. In *2018 IEEE Energy Conversion Congress and Exposition, ECCE 2018*, pages 2988–2992. Institute of Electrical and Electronics Engineers Inc., dec 2018. ISBN 9781479973118. doi: 10.1109/ECCE.2018.8558293.

[8] Samir Kouro, Patricio Cortés, René Vargas, Ulrich Ammann, and José Rodríguez. Model predictive control - A simple and powerful Method to control power converters. *IEEE Transactions on Industrial Electronics*, 56(6):1826–1838, 2009. ISSN 02780046. doi: 10.1109/TIE.2008.2008349.

[9] Tomislav Dragicevic. Model Predictive Control of Power Converters for Robust and Fast Operation of AC Microgrids. *IEEE Transactions on Power Electronics*, 33(7):6304–6317, 2018. ISSN 08858993. doi: 10.1109/TPEL.2017.2744986.

[10] Jasbir Singh Arora. *Introduction to Optimum Design*. Elsevier, 2017. doi: 10.1016/c2013-0-15344-5.

[11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. ISSN 00280836. doi: 10.1038/323533a0.

[12] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.