mathematics

Article

Artificial Neuron-Based Model for a Hybrid Real-Time System: Induction Motor Case Study

Manuel I. Capel

Department of Software Engineering, University of Granada, 18071 Granada, Spain; manuelcapel@ugr.es

Abstract: Automatic Machine Learning (AML) methods are currently considered of great interest for use in the development of cyber-physical systems. However, in practice, they present serious application problems with respect to fitness computation, overfitting, lack of scalability, and the need for an enormous amount of time for the computation of neural network hyperparameters. In this work, we have experimentally investigated the impact of continuous updating and validation of the hyperparameters, on the performance of a cyber-physical model, with four estimators based on *feedforward* and *narx* ANNs, all with the gradient descent-based optimization technique. The main objective is to demonstrate that the optimized values of the hyperparameters can be validated by simulation with MATLAB/Simulink following a mixed approach based on interleaving the updates of their values with a classical training of the ANNs without affecting their efficiency and automaticity of the proposed method. For the two relevant variables of an Induction Motor (IM), two sets of estimators have been trained from the input current and voltage data. In contrast, the training data for the speed and output electromagnetic torque of the IM have been established with the help of a new Simulink model developed entirely. The results have demonstrated the effectiveness of ANN estimators obtained with the Deep Learning Toolbox (DLT) that we used to transform the trained ANNs into blocks that can be directly used in cyber-physical models designed with Simulink.

Keywords: cyber-physical systems; neural networks; hybrid systems; automated machine learning; simulation; real-time embedded control systems

MSC: 62M45; 37M05; 65C20; 68U20

Citation: Capel, M.I. Artificial Neuron-Based Model for a Hybrid Real-Time System: Induction Motor Case Study. *Mathematics* **2022**, *10*, 3410. https://doi.org/10.3390/math10183410

Academic Editors: Nicholas Christakis, George Kossioris and Mayur Patel

Received: 19 July 2022
Accepted: 5 September 2022
Published: 19 September 2022

1. Introduction

IoT is highlighting the huge current interest in the development of autonomous driving to organize traffic in future smart cities. In this context, *automated machine learning* (AML) methods [1,2] are useful to apply machine learning end-to-end to find a consistent implementation of electric vehicles (EVs) [3] and other cyber-physical systems (CPS) [4–7]. However, they present serious application problems regarding fitness calculation, overfitting, lack of scalability, and need a huge amount of time to compute hyperparameters on which CPS are dependable. The problem is worsened if we choose an artificial neural network (ANN), and to calculate the hyperparameters [8], we follow an optimization method such as gradient descent [9,10].

Therefore, we propose in this paper to complement the computationally expensive training phase of ANNs with hyperparameter updates carried out manually, as in [11], with the help of efficient cyber-physical models' simulation tools.

The optimization algorithm normally used to train ANNs for industrial control is called gradient descent [12]. The optimization computation based on gradient descent acts as a measurement device to gauge the accuracy of the cost function calculation, which each step or iteration of the ANN parameters updates. Following this approach, evaluations of the target function results in a hyper-gradient vector, i.e., one which learns various optimized parameters to combine different levels of adaptations [13], instead of single values that are usually obtained by other methods such as the called Bayesian hyperparameter optimization (HPO) [14].

Using HPO, when the number of derivatives is large, e.g., for a deep-learning ANN with n layers, the gradient will increase exponentially by backpropagation until it finally *explodes*. We can overcome the complexity of gradient descent and backpropagation applied to CPS hyperparameters optimization by model transformation, such as the distributed optimal coordination (DOC) approach for heterogeneous linear multi-agent systems proposed in [15].

Nowadays, we can obtain the model selection gradient in a reasonable time thanks to modern parallel computers and GPU graphics cards, which are crucial to tuning hyperparameters of cyber-physical models [8]. Thanks to parallel computing recent advances, we can now handle many hyperparameters of a model by deploying gradient descent-based methods efficiently [9]; more specifically, our method for ANN applied to high-dimensional HPO problems can perform the following tasks:

1. Separately optimize the learning rate of an ANN for each iteration and layer;
2. Calculate optimal weights for the synaptic connections between neurons of different layers in the ANN;
3. Reduce the likelihood of overfitting by L2 regularization, a strategy that establishes the decay of peaky weights for each individual parameter.

Therefore, it is possible to perform the complete training procedure of an optimized cyber-physical model with an ANN, either by adaptive and offline updating of the hyperparameters: through measurements on a model of the studied CPS or after performing, interspersed with the training of the network, a manual validation from a software simulation of the cyber-physical model. This second alternative is the one we have finally adopted here.

This method is very model-specific but, in return, allows tuning of many hyperparameters of the cyber-physical model, which lays the ground for obtaining a great improvement with respect to the HPO performed so far in other papers [8,14,16], e.g., by Bayesian optimization.

Overfitting is an open problem when we try to apply ANN with a finite validation set, and the calculation of the model's hyperparameters suffers from this drawback. One possibility would be to use a different shuffling for each function we need to evaluate thus reducing the amount of overfitting. This approach has been shown to improve the generalization accuracy and recall by deploying a cross-validation strategy of the cyber-physical models. We can also use the strategy of finding stable optima instead only optima in the objective function, as noted in [16].

There are currently many machine learning problems that cannot be solved directly due to the magnitude of their scale. We will understand the term scale, in this context, as the size of the configuration space and the high computational cost of carrying out the individual evaluations of the models involved. There are currently some successes in training the neural network with small datasets and setting hyperparameter values by hand during model training [10,17]. Our approach with respect to coping with the scalability problem is to take advantage of massive parallel computing and try to fully exploit large-scale computer clusters or the thousands of SMs of GPU/CUDA multiprocessors.

In hybrid systems, the continuous behavior described by a system of differential equations associated must change as a result of the occurrence of discrete events too. Our approach gives very compact and flexible specifications for complex hybrid systems. However, there are very few tools that support this class of tools now. In our case, we hypothesize that there is no major problem in building a trained ANN, which can substitute the PID [18,19] controller of a closed loop control system, capable of reacting and producing a correct response even in the case of discrete events, i.e., messages or signals that may modify the values of the cyber-physical model's hyperparameters.

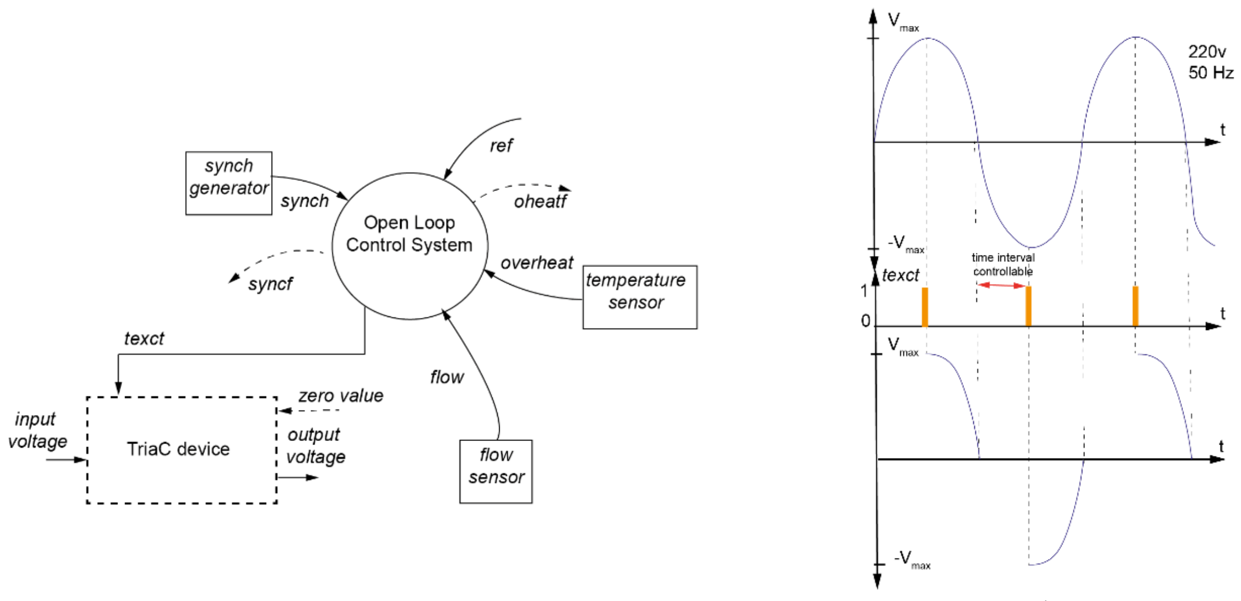
For implementation, CPS usually resorts to excellent tools such as MATLAB/Simulink © that are widely used in industry today to successfully complement machine learning methods mentioned above in IoT or Big Data applications. Currently, the use of tools that speed up the determination of the hyperparameters of ANNs is being considered a firm basis for the development of AML methods in an industrial environment and could certainly be considered a sound and less complex alternative to the development of new algorithms by only using metaheuristics that present a high complexity both in their design and in the execution time they need.

In this work, through a case study design, two research questions are answered: (1) whether an ANN model could be used to replace a PID controller for open and closed loop control systems, and (2) whether this model could be generated in a short time. To obtain the objectives, two ANN models (ff and narx) are used to generate four estimators of 2 physical variables (speed and torque) of an AC motor. Hyperparameter optimization is used with the approximate gradient method from the literature [8], which makes the gradient descent process much faster.

1.1. Real-Time Speed Regulation of an Induction Motor

As a case study, an induction motor (IM) has been modeled and controlled using simulation data and neural networks as an application of the proposed method for derivation of control applications with hybrid systems with real-time characteristics.

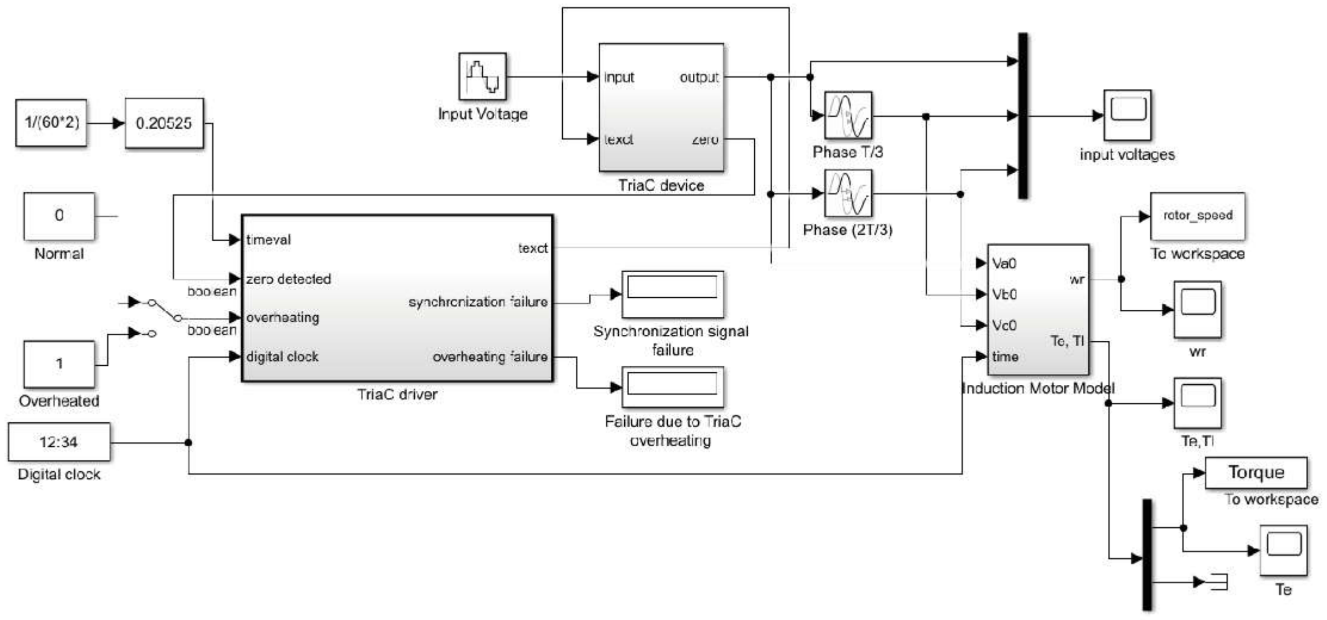
We can control the IM rotation speed by an open loop control system with real-time requirements, Figure 1a. The IM rotor speed is controlled by cutting the sine wave of the input voltage using a TriaC device, Figure 1b, whose operation is like a very fast switch. The control line of the TriaC is commanded by a synchronization signal (*synch*). This signal is *high* when the input wave passes through a zero value, resulting in the TriaC immediately stopping conducting electricity. If, after stopping conducting, the TriaC is fed with a current for several milliseconds, it will go to the saturation state by the *texct* signal. It will drive current until the input voltage passes through a zero value again. The maximum time (*timeval*) to re-excitation of the TriaC device must be calculated in real-time and at each cycle of the voltage with which it is fed. If after passing a full cycle of the input voltage, the signal *synch* is lost, i.e., the *texct* signal fails, then the *synchf* failure signal goes *high* to prevent the motor may rotate too fast. The combination of both devices, an IM and the TriaC, can serve as the basis of an industrial prototype to control or keep constant the speed of a vehicle in an automatic driving system or to maintain a constant airflow through a filter in HVAC systems, etc.



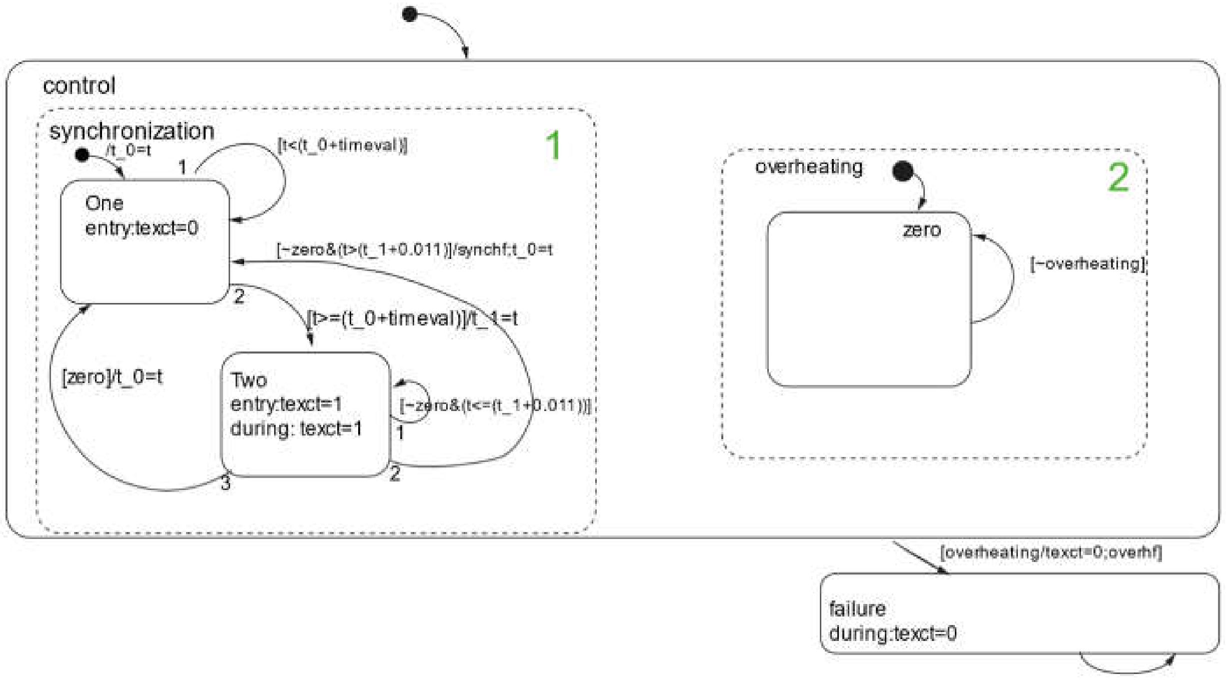
(**a**) (**b**)

Figure 1. (**a**) System context diagram. (**b**) Graphical plot of the input voltage and the excitation signal of the TriaC device.

To solve such a problem that often occurs in the manufacturing industry, we can produce a closed loop feedback system with real-time requirements by implementing a PID, necessary to maintain constant the rotor speed of an induction motor driven by a TriaC device, Figure 2a. A Simulink block can be used to implement a hybrid system that also contains discrete components, such as the TriaC driver component shown in Figure 2b.



(a)



(b)

Figure 2. (**a**) Simulink-block diagram of an induction motor controlled by a TriaC device. (**b**) Zooming the *TriaC driver* stateflow block interior.

The proposal of this paper applies, according to the approach we will detail, to the implementation of an ANN-based controller instead of a PID-based one. The case study shows how the proposed method can be applied to derive ANN estimators of rotor speed and torque of an IM that provide improved control performance of this system. We have performed a double validation of the response of the developed ANN-based estimators for speed and electromagnetic torque, considering a time interval of 50 s. of the training time. Deep Learning Toolbox (DLT) is used to transform the trained ANNs into blocks that can be directly used in cyber-physical models designed with Simulink. We can observe that the measured and estimated speed and electromagnetic motor torque have a very close trajectory in the graphs shown (performance, error histogram), as well as we can also verify that the regression study yields a good accuracy for the mentioned both engine variables estimated in the study.

1.2. Article Structure

The following sections are structured as described below. First, we present the mathematical modeling of the induction motor. In Section 3, we describe our proposed modeling approach for the design of a class of CPS, i.e., hybrid real-time systems with continuous and discrete components. We algorithmically define a new method to deploy the model, such as closed-loop control of the IM case study based on neural networks, in which we define training, testing, and validation. A reference Simulink model of the IM is used, which provides insight into the functional and dynamic aspects and the necessary training data for the ANN-based controller models proposed in this paper. In Section 4, two motor speed and torque estimators are obtained, each with two classes of ANNs (ff and narx), which are applied to solve a problem consisting of reliably maintaining a constant speed and electromagnetic torque of the IM of the case study. Conclusions and current lines of work are presented in the last section.

2. Induction Motor Modeling

The simulation of the induction motor (IM) in Simulink, which has been used as a case study here, can be downloaded at the address https://lsi2.ugr.es/~mcapel/miscelanea/motor/ (accessed on 20 May 2022) (see Supplementary Materials). The prototype code has been structured in three separate blocks, corresponding to the transformation between the reference system of magnetic fluxes, currents, and voltages in the motor, resolution of the simulation, and return to the standard three-phase reference system, as shown in Figure 3a.

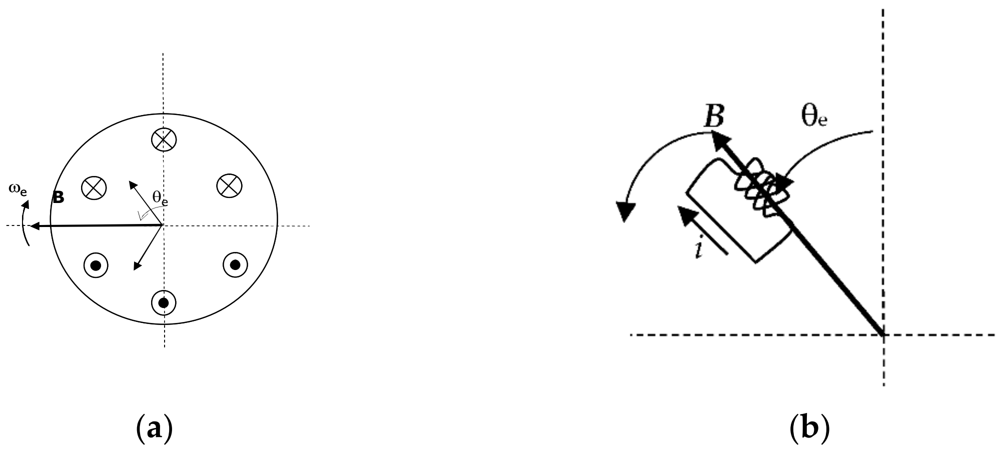


Figure 3. In this figure, we can observe: (**a**) the motor stator winding and the rotating magnetic field it produces; (**b**) the induction motor model with only one rotating coil at the speed ω_e of the rotating field $\boldsymbol{B}$.

2.1. Physical Modeling of an Induction Motor

The most difficult component to model an induction motor (IM) is the IM drive itself, as specialized Simulink blocks that include differentiation operators are needed or need to be developed manually, as shown in Section 2.3. The other devices in the IM model can be modeled by means of simple switches or a combination of them.

An IM works according to the physical principle of mutual induction between electrical circuits traversed by a varying magnetic flux. Applying Faraday's law, which is given by the following equation, where N is the number of turns of wire and Φ_B is the magnetic flux through a single loop, we can obtain the relation between electromagnetic force (ε) and the change of magnetic flux Φ_B through the winding of a motor as follows,

$$\varepsilon = -\frac{d}{dt}(N \cdot \Phi_B), \tag{1}$$

i_{reel} conducted by the stator circuit, i.e., it turns out to be independent, for example, on the number of poles of the motor. Therefore, we will assign a self-induction constant L to any circuit affected by magnetic induction, according to the following equation,

$$N \cdot \Phi_B = L \cdot i_{reel} \tag{2}$$

To obtain a rotating magnetic field $\vec{B}$ (or $\boldsymbol{B}$) in the stator of an IM, as shown in Figure 3a, three windings are necessary for the stator nucleus, each of which conducts current with a voltage phase difference of $2\pi/3$ rad. from the next. An important parameter of induction motors and thus necessary for our target model to learn is what we call the synchronous speed ω_e of the motor, which is the rotational speed of the magnetic field $\boldsymbol{B}$ produced by the triple winding of the stator.

A three phases model of a dynamic IM can be found by assuming balanced voltages (in the phase of $2\pi/3$ rad) and constant inductances. Under these conditions, the magnetic field vector $\boldsymbol{B}$ in the stator describes a circle in the plane with speed ω_e. A mathematical model with a rotating vector $\boldsymbol{B}$ identical to the above can be obtained with single current i instead of using three sinusoidal currents in phase. This model would only define one coil rotating at the same speed as vector $\boldsymbol{B}$ (Figure 3b) and conducting a direct current (DC). The problem is that with this model, the inductances' mutual couplings between the three stator windings would not be considered. Therefore, a minimum of 2 coils are needed to obtain a dynamic model of the IM, as Figure 4 shows.

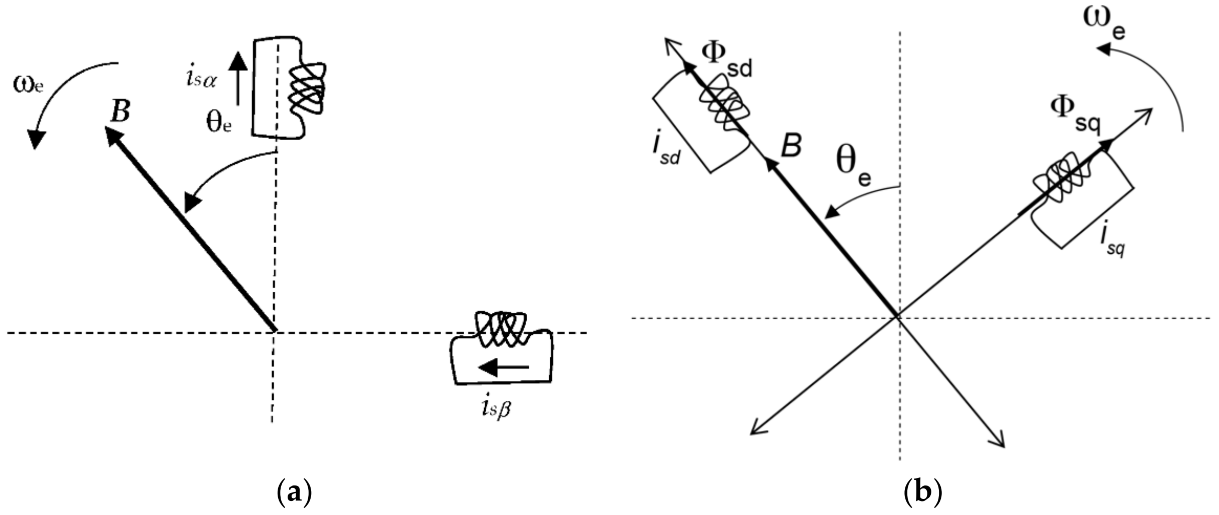


Figure 4. The reference systems that simplify the equations of the magnetic induction in the AC motor model: (**a**) $\alpha\beta$ or *stationary*; (**b**) *qd* or *rotative*.

Complex vector space can be represented using a reference frame of only two orthogonal axes, according to the two-axis theory [20], so that in our study the motor model could be considered equivalent to a two-phase machine, which is a model that reduces the number of equations and greatly simplifies the control design of an IM.

2.1.1. Stationary Reference System (Alpha and Beta Coordinates)

In this case the motor is modeled by placing two coils forming an angle of $\pi/2$ rad (Figure 4a) and each one of them drives a sinusoidal current with a phase difference of $\pi/2$ rad. The first coil (or coil-α) will conduct the first phase of the current ($\varphi = 0$) and it is fixed to the ordinate axis. The second coil (or coil-β) will conduct the second phase of the current ($\varphi = \pi/2$) and is considered fixed to the x-axis. The named $\alpha\beta$-reference system reduces the IM modeling problem to a simpler electromagnetic model with only two phases.

2.1.2. Rotative Reference System

Combining the single rotating coil system and the stationary reference system yields a new system consisting of two rotating coils, Figure 4b, which allows to obtain a rotating vector $\boldsymbol{B}$ and allows the modeling of the couplings between the inductances of the coils. In addition, the currents conducted by the coils could be direct (DC).

In this model, the axis of reference, which is solidary with the coil driving the phase α is called the d-axis (from the *direct* axis), and the axis corresponding to the coil β is called the q-axis (from the quadrature axis). Where θ_e is the angle between the α-axis of the stationary reference attached to the stator and the d-axis of the rotative reference frame, we will always consider the rotor angle θ_e as just another state of the IM model and, therefore, a value that can be calculated.

2.1.3. Motor Shaft Rotation

To form the magnetic pole pairs that will produce the movement of the motor shaft, the rotor winding must also conduct a current to generate magnetic flux. If we short-circuit the rotor winding (for example, using a *squirrel cage* winding [21]), then a current is induced in it, which produces an electromagnetic force. The motor will start to rotate at the speed ω_r due to the electromagnetic force generated by the change of direction of the magnetic field $\boldsymbol{B}$, which rotates at the synchronous angular speed ω_e resulting from the stator winding.

On the other hand, according to Faraday's law (1), only if the magnetic flux Φ varies with time will an electromagnetic force be generated, which translates into a momentum capable of moving a load. Such a change may be due to a change in the amplitude of the magnetic flux Φ or the electric circuit representing the rotor winding that cuts the magnetic flux and changes its angle of incidence. According to this observation, it will be the case that if the rotor were to rotate at the same speed as the rotating magnetic field of the stator, i.e., $\omega_r = \omega_e$, then no electromagnetic force would be induced, nor would any momentum be created. If, on the other hand, there were a small difference between the rotor angular velocity ω_r and the $\boldsymbol{B}$ speed ω_e induced by the stator, then an electromagnetic force would be generated and, consequently, create a momentum that can be converted into mechanical torque. The difference between both rotational speeds (ω_e and ω_r) is another important parameter of induction motors, which is called *slip*, Table 1 shows the values assumed by these variables during a random run of the model.

Table 1. Synchronous and rotor speeds.

Slip Feature of 2-Pole IM	
ω_e: stator synchronous speed (input variable)	314.2 rad/s
ω_r: rotor angular speed (output variable)	311.2 rad/s
s: slip (IM parameter)	0.01 rad/s

2.1.4. Transformation of the 3-Phase Reference Model to the Rotating Reference System

In three-phase symmetrical machines, the direct and quadrature axis currents (as well as the other variables) that are defined in the rotating reference frame represent two fictitious components of the stator and rotor currents components. The components of the three-phase reference frame currents can be transformed into the direct and quadrature axis components by means of a linear transformation and vice versa.

It is necessary to find the transformations that allow passing the currents $\vec{i}$ (i_a, i_b, i_c), voltages $\vec{v}$ (v_a, v_b, v_c), and magnetic fluxes $\vec{\Phi}$ (Φ_a, Φ_b, Φ_c) to the stationary 2-phase reference frame: $\vec{i}_{\alpha\beta}$, $\vec{v}_{\alpha\beta}$, and $\vec{\Phi}_{\alpha\beta}$, then, to the rotating reference system: $\vec{i}_{\mathrm{dq}}$, $\vec{v}_{\mathrm{dq}}$, $\vec{\Phi}_{\mathrm{dq}}$. Trivially, from a mathematical point of view, an expression given as a function of the coordinates *dq* can be directly transformed into an expression in coordinates $\alpha\beta$ if the θ_e angle always maintains a constant value $\theta_e = 0$. Otherwise, we will therefore have to apply the transformation of reference systems ($abc \to dq$), expressed by the following *T* matrix transformation:

$$\vec{f}_{dq} = T_{dq}(\theta)\cdot\vec{f}_{abc} \tag{3}$$

where $\vec{f}_{dq}$ is a vector representing the voltage, current or magnetic flux of the stator or rotor (we will use different variables for each of these motor elements) of the IM model. This is a transformation from a three-phase system (*abc*) into a two-phase system (*dq*). However, the *d* and *q* axes may be insufficient to describe the above physical variables if the currents are not balanced or if induced magnetic fluxes occur, causing the neutral wire of the stator to drive current, i.e., that $v_0 \neq 0$. Because of this, the zero component is added to the physical magnitude $\vec{f}$, and we will therefore denote the reference system as *dq0* from now on. The transformation matrix of the three-phase reference system (*abc*) into the stationary reference frame ($\alpha\beta 0$) is given in [3].

$$T_{\alpha\beta 0} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1/\sqrt{3} & 1/\sqrt{3} \\ 1 & 1 & 1 \end{pmatrix}$$

The inverse transformation matrix (from system $\alpha\beta 0 \to abc$) would be:

$$T^{-1}_{\alpha\beta 0} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & -\sqrt{3}/2 & 1/2 \\ -1/2 & \sqrt{3}/2 & 1/2 \end{pmatrix}$$

To transform any of the above magnitudes, for example, voltage, to the rotating reference system, the following equations can be used:

$$\begin{cases} V_{qs} = V_\alpha cos\theta_e - V_\beta sen\theta_e \\ V_{ds} = V_\alpha sen\theta_e + V_\beta cos\theta_e \end{cases}$$

The inverse transformation would be given by the equations:

$$\begin{cases} V_\alpha = V_{qs}\cos\theta_+ + V_{ds} sen\theta_e \\ V_\beta = -V_{qs} sen\theta_e + V_{ds}\cos\theta_e \end{cases}$$

2.2. Physical Model of the IM in the Rotative Reference System

The IM model can be derived from the equivalent electrical circuit (Figures 5 and 6). The differential equations describing this model expressed as a function of the magnetic linkage *Fij* (see definition in Table 2) are as follows:

$$\frac{dF_{qs}}{dt} = \omega_b\left[v_{qs} - \frac{\omega_e}{\omega_b}F_{ds} + \frac{R_s}{\chi_{ls}}\left(\frac{\chi^*_{ml}}{\chi_{lr}}F_{qr} + \left(\frac{\chi^*_{ml}}{\chi_{ls}} - 1\right)F_{qs}\right)\right] \tag{4}$$

$$\frac{dF_{ds}}{dt} = \omega_b\left[v_{ds} + \frac{\omega_e}{\omega_b}F_{qs} + \frac{R_s}{\chi_{ls}}\left(\frac{\chi^*_{ml}}{\chi_{lr}}F_{dr} + \left(\frac{\chi^*_{ml}}{\chi_{ls}} - 1\right)F_{ds}\right)\right] \tag{5}$$

$$\frac{dF_{qr}}{dt} = \omega_b\left[-\frac{(\omega_e - \omega_r)}{\omega_b}F_{dr} + \frac{R_r}{\chi_{lr}}\left(\frac{\chi^*_{ml}}{\chi_{ls}}F_{qs} + \left(\frac{\chi^*_{ml}}{\chi_{lr}} - 1\right)F_{qr}\right)\right] \tag{6}$$

$$\frac{dF_{dr}}{dt} = \omega_b\left[\frac{(\omega_e - \omega_r)}{\omega_b}F_{qr} + \frac{R_r}{\chi_{lr}}\left(\frac{\chi^*_{ml}}{\chi_{ls}}F_{ds} + \left(\frac{\chi^*_{ml}}{\chi_{lr}} - 1\right)F_{dr}\right)\right] \tag{7}$$

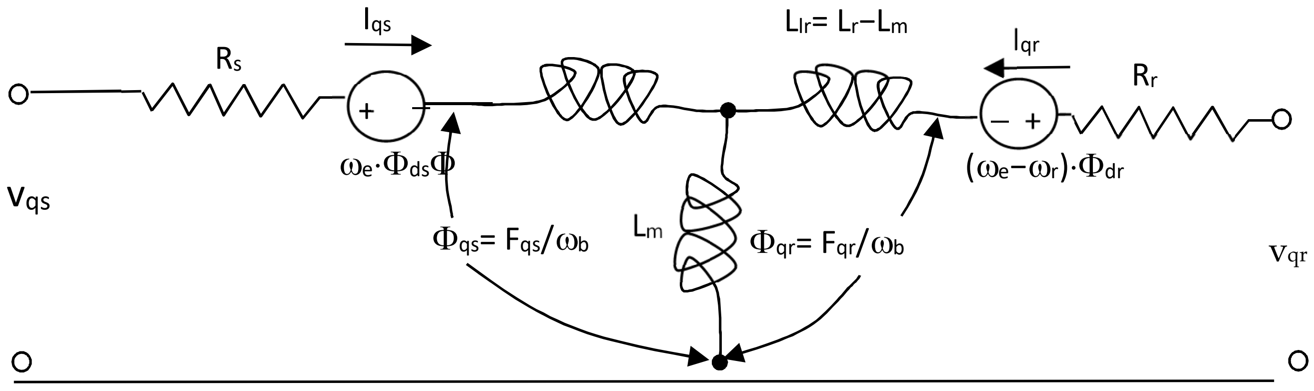


Figure 5. Dynamic equivalent circuit of an IM along the *q*-axis of the rotating reference system.

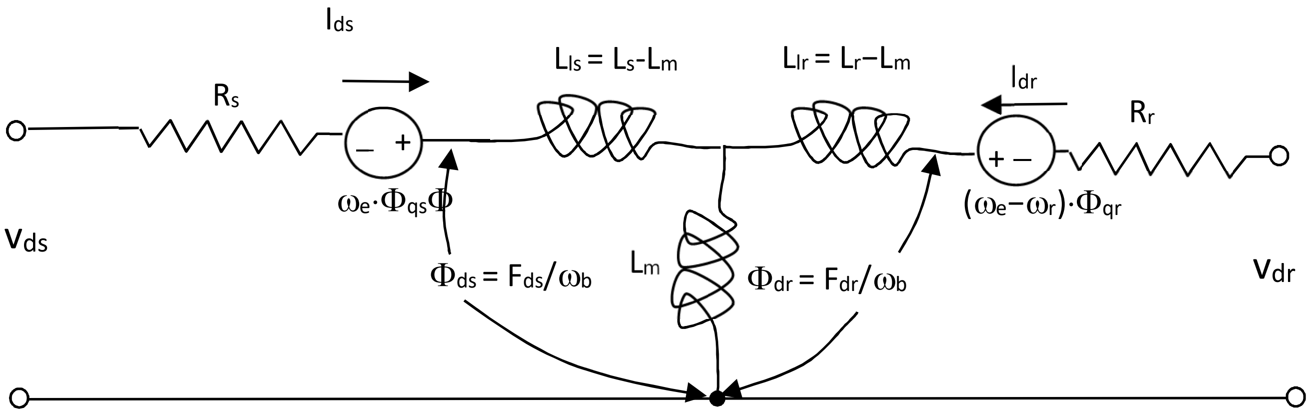


Figure 6. Dynamic equivalent circuit of an IM along the *d*-axis of the rotating reference system.

Table 2. Physical constants and variables of the induction motor model.

Relevant Observables and Magnitudes of the Physical System under Study	
d: direct axis of the rotating reference system	χ^*_{lm} = 1/(1/χ_{ls} +1/χ_{lr} + 1/χ_m): total reactance with the loses for magnetizing (χ_m)
q: quadrature axis of the rotating reference system	i_{qs}, i_{ds}: currents of the *q* and *d* stator axis
s: subindex for the stator variable	i_{qr}, i_{dr}: currents of the *q* and *d* rotor axis
r: subindex for the rotor variable	*p*: number of poles of the motor
F_{ij} = Φ_{ij}, ω_b: magnetic linkage, where *i* = *q* or *d* and *j* = *s* or *r*	*J*: inertia momentum
v_{qs}, v_{ds}: stator voltages	M_e: motor's electromagnetic torque (output variable)
v_{qr}, v_{dr}: rotor voltages	M_l: load torque (input variable)
R_r, R_s: rotor and stator resistors	ω_e: stator synchronous speed (input variable) of ***B***
χ_{ls}: stator reactance ($\omega_e \cdot L_{ls}$)	ω_b=2·π·f_b: angular speed corresponding to the electric frequency of feeding voltage
χ_{lr}: rotor reactance ($\omega_e \cdot L_{lr}$)	ω_r: rotor angular speed (output variable)

As can be seen in Equations (6) and (7), in the IM model, voltages v_{qr} and v_{dr} are identically equal to zero in the rotor since we assume a short-circuited (squirrel cage) rotor winding; and therefore, the right-hand side of the equivalent electrical circuits would have zero voltage. The constants and variables of the above system of linear differential equations are defined in Table 2.

Simulink Model of the Induction Motor Model

A new block has been implemented using Simulink from the physical model, given by the differential Equations (4)–(7) of the induction motor. In the first column of the Simulink model (Figure 7), the blocks corresponding to the differential equations of the four magnetic linkages F_{ij} appear since they are needed to calculate the other output variables. Each of these equations could have been implemented using the Simulink block intended for modeling systems of equations in state-space block notation, but for reasons of design flexibility, discrete user blocks have been used instead.

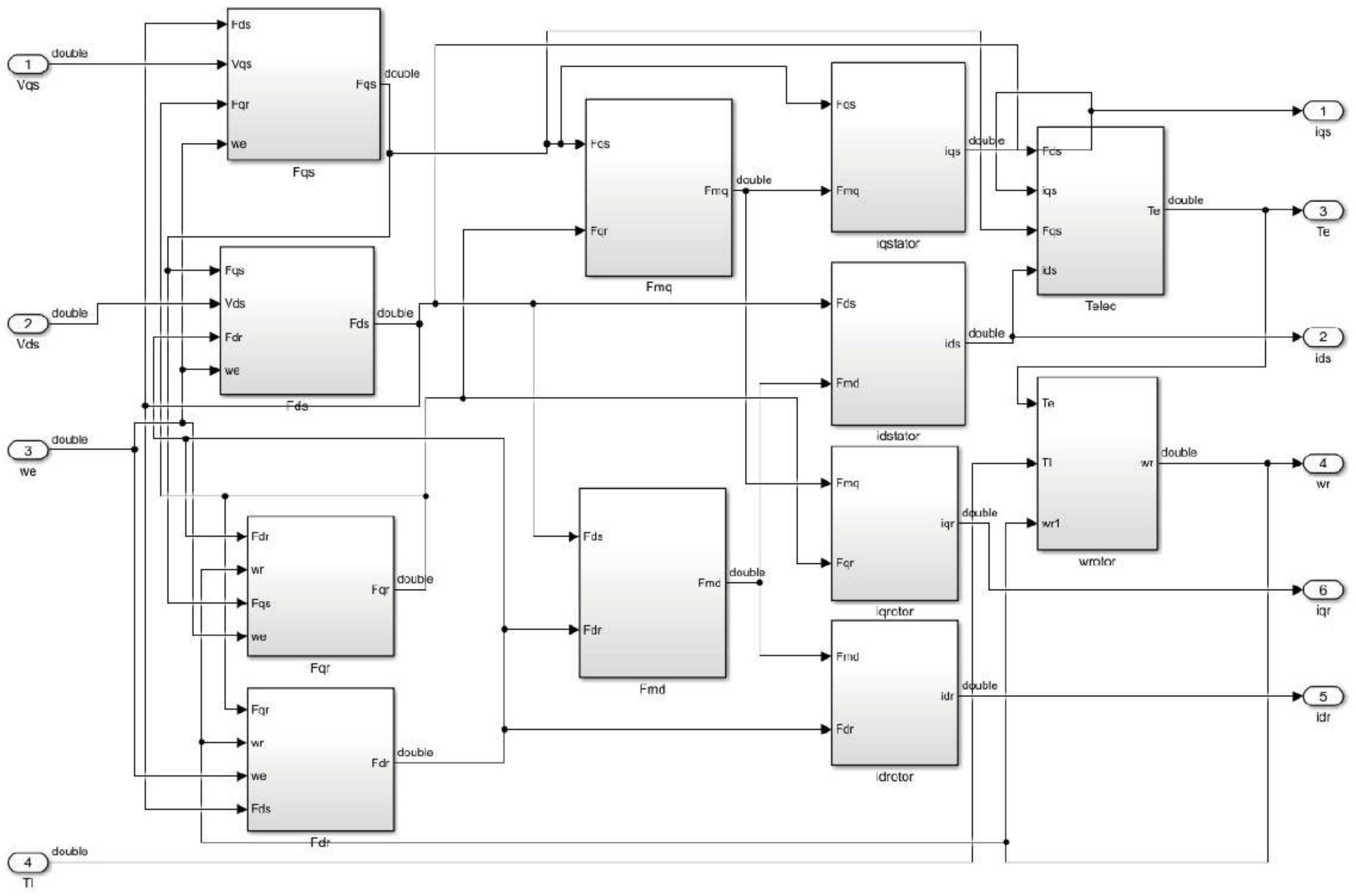


Figure 7. Simulink block representing the dynamic model of an induction motor.

Figure 8 shows the interior design of one of the blocks of the first column—the one that calculates the magnetic linkage F_{ds}.

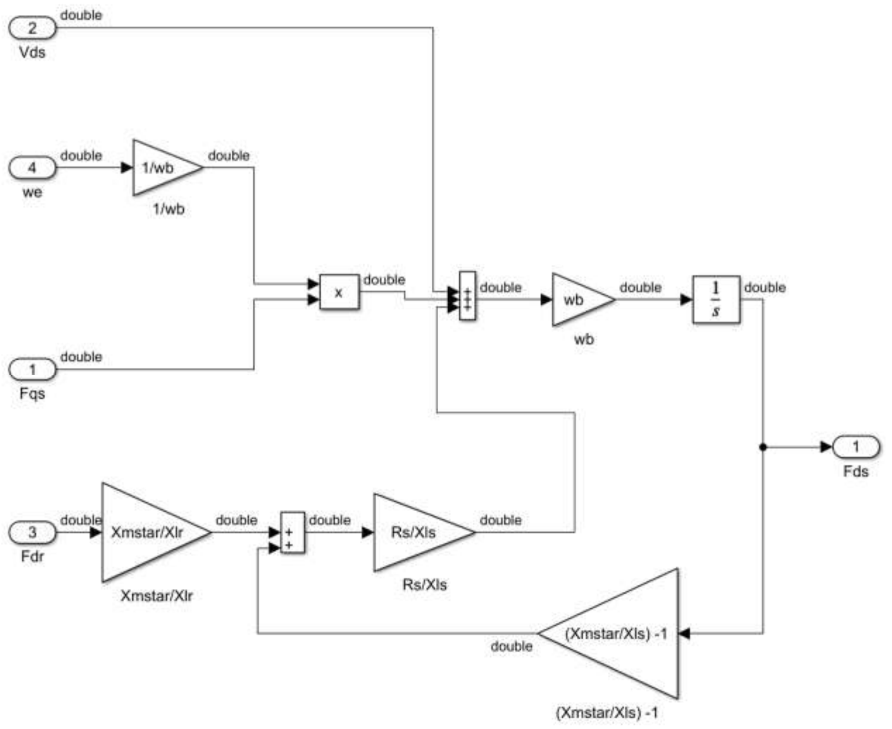


Figure 8. Simulink block that solves the differential equation of the F_{ds} flow (Xmstar is the total magnetizing reactance χ^{*}_{lm}).

2.3. Calculation of the Electromagnetic Torque and Mechanical Power Generated by the Motor

The most important function of the MI is to produce mechanical torque on the motor shaft. The electrical power input to the motor P_{input} in the three-phase reference system can be obtained by evaluating the following expression, which is a sum of terms of products of the voltage and current values of each of the phases of the motor input current:

$$P_{input}= v_{as}\cdot i_{as}+v_{bs}\cdot i_{bs}+v_{cs}\cdot i_{cs}+v_{ar}\cdot i_{ar}+v_{br}\cdot i_{br}+v_{cr}\cdot i_{cr} \tag{8}$$

The expression of Equation (8) above can be easily obtained by simply applying the well-known electric power formula: $P_e = \sum_{i,j} v_{ij}\cdot i_{ij}$; where $j = s$ (stator) or r (rotor) and $i = a, b, c$, to the equivalent electric circuit of the induction motor (Figure 9), which we can consider as an accurate model for calculating the input power of the motor. The input power is expended on four things: electromagnetic energy, mechanical energy, dissipation by losses in resistors and inductances, and the movement of the load, although most of the electrical power generated by the motor is converted into mechanical torque.

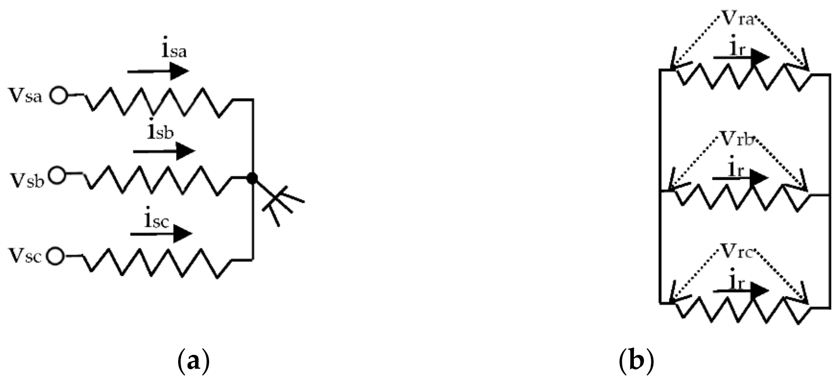


Figure 9. Equivalent electrical circuits for (**a**) stator and (**b**) rotor windings.

In order to comfortably use the input power equation, we have obtained above, we have to transform it according to our rotating reference frame *dq0*, using for this purpose the inverse transformation given by $T^{-1}{}_{\alpha\beta 0}$, and after conveniently manipulating Equation (8), we obtain the following expression for the input power expressed in the reference frame *dq0*,

$$P_{input} = \frac{2}{3}\left(v_{ds}\cdot i_{ds} + v_{qs}\cdot i_{qs} + 2v_{0s}\cdot i_{0s} + v_{dr}\cdot i_{dr} + v_{qr}\cdot i_{qr} + 2v_{0r}\cdot i_{0r}\right) \tag{9}$$

The currents produced in the zero phase of the stator, and consequently the magnetic fluxes induced in this phase, do not contribute to the mechanical power of the motor. Moreover, as previously commented, if we consider that the rotor winding is short-circuited, then the voltages v_{qr}, v_{dr}, are zero, so that from Equation (9), we will only consider the first two terms of the sum. On the other hand, if we substitute the voltages by their equation as a function of the magnetic induction and the stator current intensities, Equation (9) would be re-written as follows:

$$P_{input} = \frac{3}{2}\left(r_{ds}\cdot i_{ds}^2 + \omega_b\phi_{qs}\cdot i_{ds} + \frac{d}{dt}\phi_{ds}\cdot i_{ds} + r_{qs}\cdot i_{qs}^2 - \omega_b\phi_{ds}\cdot i_{qs} + \frac{d}{dt}\phi_{qs}\cdot i_{qs}\right) \tag{10}$$

In the above equation there are three different types of generated power. The first type has the form ri^2 and corresponds to the power dissipated in the motor windings. The second type has the form $\omega\cdot\phi_i$ and represents the electrical energy converted into mechanical energy. The third type is of the form $\frac{\mathrm{d}}{\mathrm{dt}}\phi\cdot i$ and takes into account the energy exchange between the magnetizable parts of the machine and the windings. Therefore, the total mechanical power will be given by the following equation for an IM with *p* poles,

$$P_{mech} = \frac{3}{2}\left(\frac{p}{2}\right)\left(\omega_b\cdot\phi_{qs}\cdot i_{ds} - \omega_b\cdot\phi_{ds}\cdot i_{qs}\right) \tag{11}$$

and the magnetic flux $\phi_{qs,ds}$ only depends on the angular speed and the magnetic linkage $F_{ij} = \omega_b\cdot\Phi_{ij}$. The mechanical torque transmitted by the motor shaft would be equivalent to the electrical torque generated by the motor, $P_{mech} = \omega_b\cdot M_e$, then we can obtain,

$$M_e = \frac{3}{2}\left(\frac{p}{2}\right)\frac{1}{\omega_b}\left(F_{qs}i_{ds} - F_{ds}i_{qs}\right) \tag{12}$$

from the magnetic linkages F_{ds}, F_{qs}, and currents, which are obtained by solving a system of differential linear equations, graphically represented by (Figure 7), with concrete values of *p* (the number of poles) and ω_b (stator speed) as the input data to the induction motor model.

The angular speed ω_r of the rotor can also be calculated since the mechanical load torque $M_{l,}$ and inertia *J* are also parameters of the IM model. As the above Equation (12) shows, the electrical torque M_e (and the angular rotor speed ω_b) depend on the number of poles of the rotor winding, contrary to what happens with the magnetic fluxes.

3. Modeling Method

In the proposed method, we will use different ANN classes [22,23] to design a hybrid real-time system with continuous and discrete components. We use a typical design of a neural network, in which we define training, testing, and validation (Figure 10). The output generated or "measured" from the IM Simulink model, which represents the functional and dynamic aspects of the model, are the training data.

During the execution of the IM drive implemented with Simulink, the main variables of the CPS acquire values that are used to train the ANN. The MATLAB Deep Learning Toolbox (DLT) allows us to generate Simulink blocks that implement different types of neural networks (feed-forward, narx, . . .).

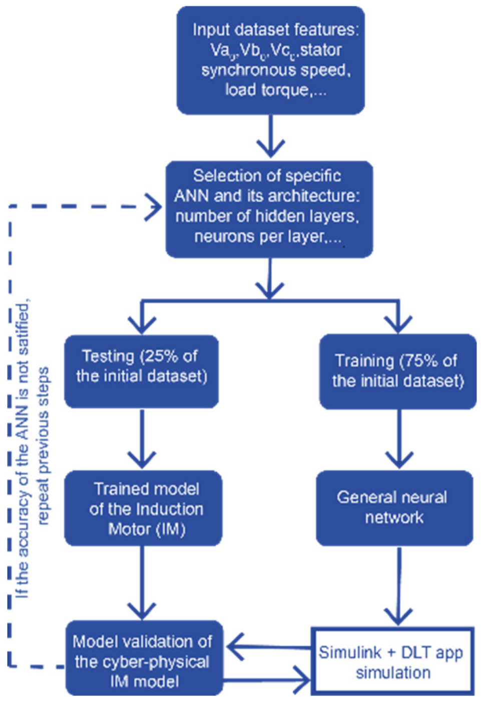

Input dataset features: Va, Vb, Vc, stator synchronous speed, load torque,...
Selection of specific ANN and its architecture: number of hidden layers, neurons per layer,...
Testing (25% of the initial dataset)
Training (75% of the initial dataset)
Trained model of the Induction Motor (IM)
General neural network
Model validation of the cyber-physical IM model
Simulink + DLT app simulation
If the accuracy of the ANN is not satified, repeat previous steps

Figure 10. Diagram of the approach to deploy the general model of a cyber-physical system; there is only one neural network that learns from input data and outputs the next value.

A Simulink block implementing a trained ANN can replace any PID controller in a closed-loop control system, such as those typically used in industrial systems to keep the output signal constant and produce a predictable system response, even in the case of disturbances in the input signals or noise. We can see the advantages of the mentioned substitution in [18].

3.1. Registry

The training data were obtained by simulation using the Simulink model of the IM drive. The problem of data logging, although facilitated by the assistance offered by the Simulink software, requires the application of algorithms such as the classical ICP or some of its adaptations [24].

The proposed method starts from N sets of *point clouds*: $\{(v_a, v_b, v_c, i_a, i_b, i_c, M_l, \omega_e)\}_{i=1}^{N}$, whose points have information of a 3D static property, i.e., the voltages: v_a, v_b, v_c and currents: i_a, i_b, i_c of the stator, and of two dynamic properties, which have been chosen to be the load torque M_l and the ω_e synchronous angular speed of $\boldsymbol{B}$ in the stator (see Table 2).

The point clouds for ANN training can be obtained in several ways: by direct measurement of the physical device (IM) using a rotating torque-meter or obtained directly (off-line) from an experimental database for three-phase induction motor rotor fault detection and diagnosis [25], or they can be obtained by simulating the IM drive with Simulink, which is the option chosen in this study. The acquisition of the training data must be repeated several times, and outliers must be eliminated. In the end, the objective of the first stage of the proposed method is to obtain a realistic point cloud in which the evolution of the rotor's angular speed ω_r and the rotor's electromagnetic torque M_e in the real physical system can be visualized. On the other hand, the objective model sought by the method, to accurately predict the values of the above variables, must be able to react and self-stabilize against variations of input voltages, or changes of the load torque M_l and synchronous speed ω_e, to achieve this the IM drive Simulink model proves to be of great help.

The training data have been collected assuming initially a synchronous reference speed $\omega_e = 2\pi \times f_b$, where f_b is the frequency of the feeding voltage (=100 s^{-1}.) in the stator winding. M_l is the load torque, whose values are in the range [0.0 … 330.0] Nm during the simulation. The collected data from the Simulink model are the voltages and currents in the three phases of the named *abc* reference system. Figure 11 shows the reference rotor speed ω_r, which is assumed to change over the simulation time. The simulation time was 50 s. The simulation was done with a sample time of 0.0005 s and the size of data points is 250,000. The rotor speed ω_r has been stored in the MATLAB workspace as a time series represented by a table with the same time step as the simulation sampling time.

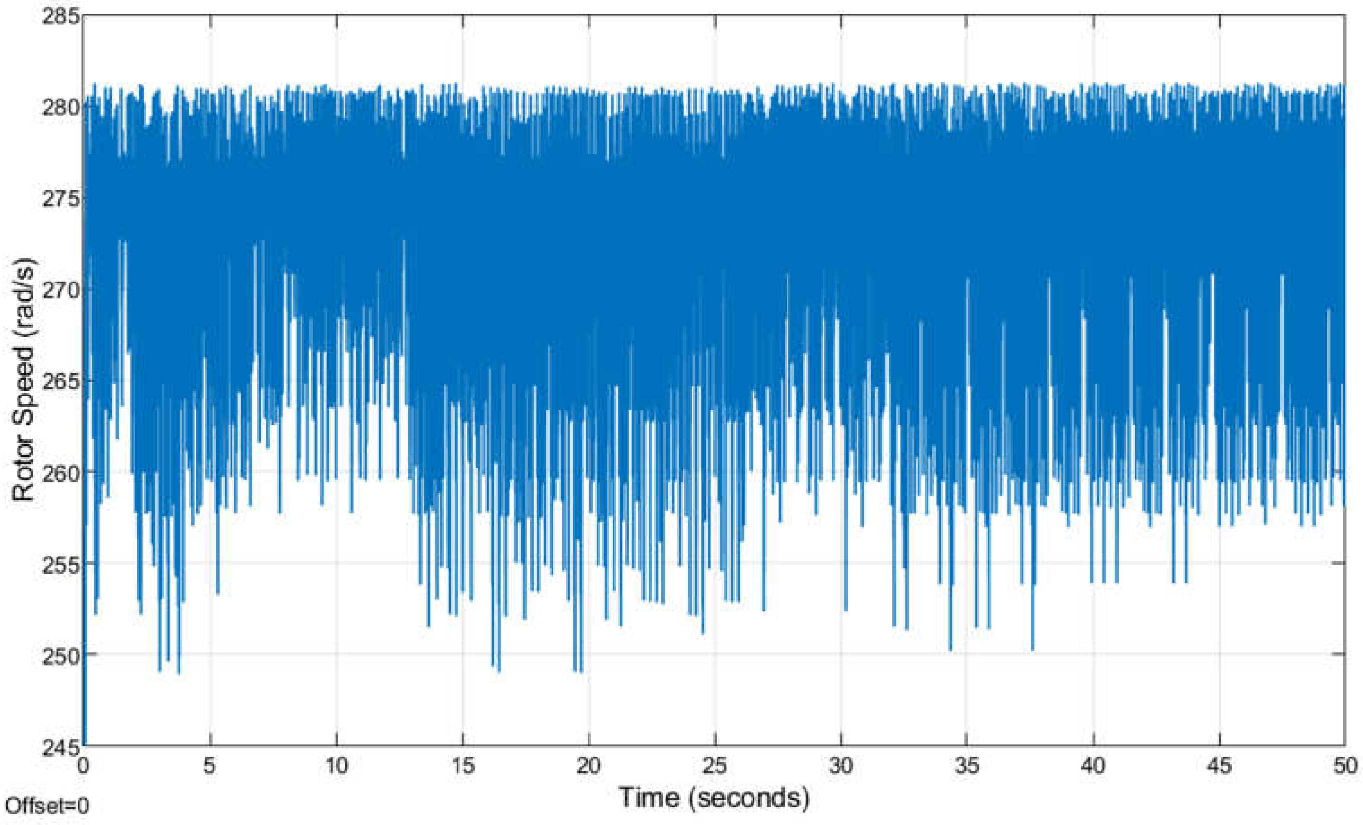


Figure 11. Applied reference rotor speed. Training data are collected from the Simulink model of the IM drive.

3.2. Design of the ANN and Selected Hyperparameters

The type of ANN is very important to get an accurate estimation of the electromagnetic torque and speed of the IM. The DLT toolbox of MATLAB has been used to implement the ANN deployed in the study. The ANN hyperparameters related to the structure (number of neurons and hidden layers ...) have been chosen by trial and error; those related to how the network is trained (backpropagation step, activation function, epochs ...) are given by the selected training function and, usually, can be adjusted manually. A shallow multilayer neural network with two hidden layers has been initially defined, Figure 12a. The training function of both ANN has been chosen as the Levenberg-Marquardt ('trainlm') one, but any other could have been chosen instead.

The objective function selected was the mean squared error (MSE). Of course, the number of hidden layers and the number of neurons in each layer can be tuned to obtain, with the maximum possible performance, the estimated speed within a time window (50 s). We have selected here 20 and 10 neurons for the first and second hidden layers, respectively.

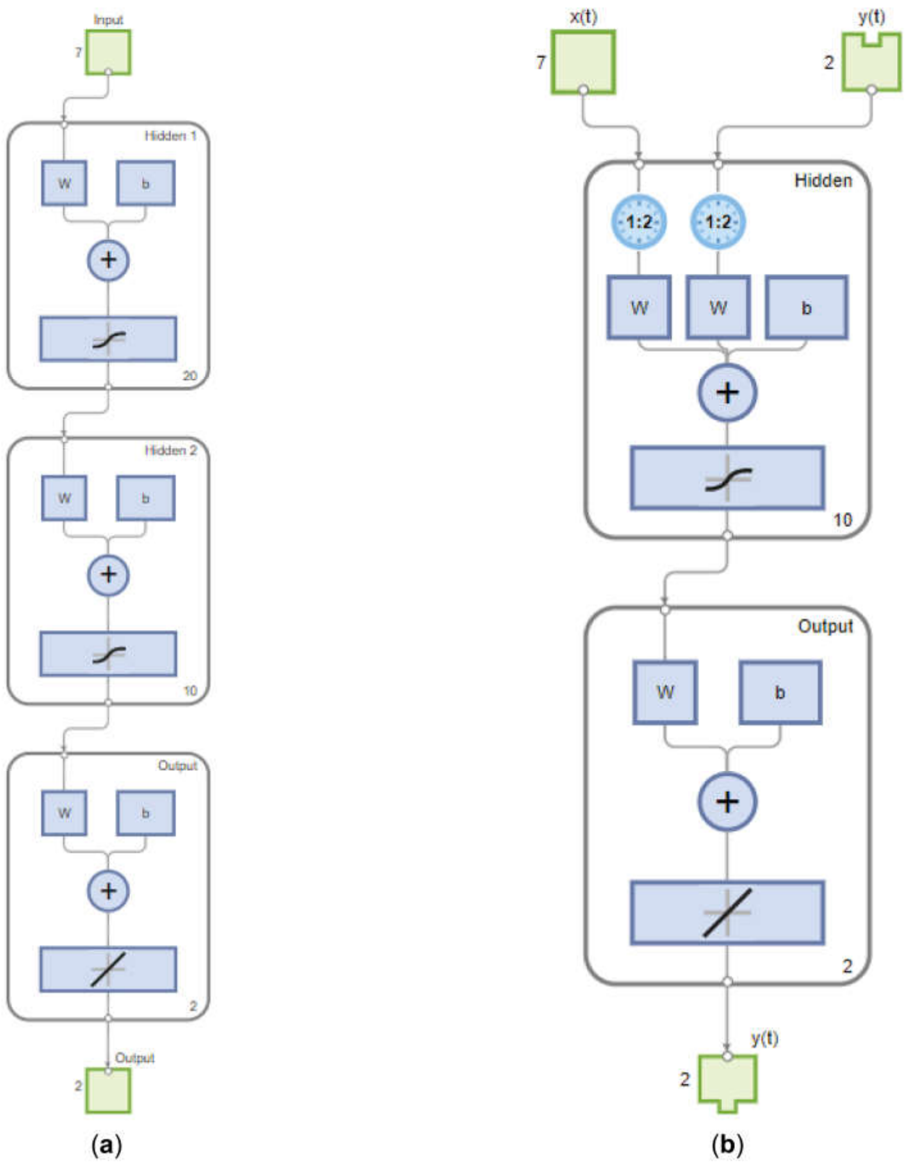


Figure 12. Architectures of the neural networks deployed in the study. (**a**) Feed-forward neural network with two hidden layers; (**b**) Narx implemented with a two-layer feed forward network.

In the second part of the study, a non-linear autoregressive with exogenous input neural network has been used, Figure 12b, i.e., a two-layer feed-forward narx, with ten neurons and a sigmoid transfer function (depicted as one Simulink block with curved line), which takes an input data matrix and returns another one where each column vector contains a single value '1', with all other elements equal to '0'. In the output layer, one linear transfer function (Simulink block with an oblique straight line) has been defined.

3.3. Training of the ANN

The main objective of ANN training is to minimize the objective function $F(x, \Pi_i)$ iteratively by fitting a set $\Pi_i = \{\alpha,\beta,\gamma,\ldots\}$ of hyperparameters with respect to K training samples, which have been initially chosen to be a subset of the point clouds yielded by the prior registry stage, i.e.,

$$x = \left\{v_a^i, v_b^i, v_c^i, i_a^i, i_b^i, i_c^i, M_l^i, \omega_e^i\right\}_{i=1}^{K} \tag{13}$$

where v_x, i_x, M_l, ω_e are the input voltages and currents in the three-phase reference system, the load torque, and the synchronous speed, respectively.

This process can be expressed by the following equation, $min_{\Pi_i} F(x, \Pi_i)$, such that

$$F(x, \Pi_i) = l.r.\left(\left\{v_a^i, v_b^i, v_c^i, i_a^i, i_b^i, i_c^i, M_l^i, \omega_e^i\right)\}_{i=1}^K, \Pi_i\right) + r(\Pi_i) \tag{14}$$

where F is the objective function, *l.r.* represents a linear relationship between each set of training samples and the hyperparameters Π_i, and the function $r(\cdot)$represents the penalties that the application of each set of hyperparameter values may incur.

The optimization of the objective function F can be performed by several techniques: reinforcement learning, heuristics, and gradient descent, ... In our method, we have chosen the latter one because the functions describing the cyber-physical model are all differentiable. F maintains the same properties, so that the calculation of $min_{\Pi_i} F(x, \Pi_i)$ does not usually diverge if gradient descent application is chosen.

The training of each of the ANNs used in the study has been performed offline, using the recorded currents and voltage data during a representative run of the IM drive model to save implementation time. The rotor speed ω_r was then predicted ('*estimated speed*') by training first the ff ANN and then the narx ANN. The same process has been carried out for the estimation of the electromagnetic torque M_e.

3.4. Validation of the ANN

In order to obtain a useful objective function adapted to our case study, we will define a cost function to be optimized and build the complete performance evaluation model, for which we have selected the MSE based on four key factors: v_x, i_x, ω_r and the M_e, (see definitions in Table 2) affecting the reliability and efficiency of the IM model.

Therefore, we can interleave the dimensions of the model ($\overrightarrow{v_x}$, $\overrightarrow{i_x}$, M_e, ω_r) to obtain a multidimensional evaluation of performance, such that the MSE can guide the iterations during the application of the gradient descent technique,

$$I_{k_x} = \frac{\sqrt{\sum_{i=1}^K \left(\overrightarrow{i_{x_i}} - \overrightarrow{i_{x_i}^{ref}}\right)^2}}{K} \tag{15}$$

$$E_{k_x} = \frac{\sqrt{\sum_{i=1}^K \left(\overrightarrow{v_{x_i}} - \overrightarrow{v_{x_i}^{ref}}\right)^2}}{K} \tag{16}$$

$$M_{e_k} = \frac{\sqrt{\sum_{i=1}^K \left(M_{e_i} - M_{e_i}^{ref}\right)^2}}{K} \tag{17}$$

$$W_{r_k} = \frac{\sqrt{\sum_{i=1}^K \left(\omega_{r_i} - \omega_{r_i}^{ref}\right)^2}}{K} \tag{18}$$

The reference values of the four key factors for MSE calculation are the measured values of currents, voltages, torque, and synchronous speed in each one of the K training samples. We can find the optimal model by minimization of

$$F(\Pi) = \log\left(\alpha \cdot E_{k_x} + \beta \cdot I_{k_x} + \gamma \cdot M_{e_k} + \delta \cdot W_{r_k}\right) \tag{19}$$

We consider that the natural logarithm can substitute the square root. In the latter equation α, β, γ, and δ denote the weight hyperparameters, and E_{kx}, I_{kx}, M_{ek}, W_{rk} represent the performance factors of input voltages, currents, torque, and motor speed, respectively.

First, we create the network by executing '*name* = feedforwardnet ([10, 20])' of DLT; i.e., we create a two-layer feed-forward network with 10 and 5 hidden neurons, respectively. In the second part of the study, we repeated the process by creating a narx network by issuing the command *name* = narxnet(10). By executing the command 'view (*net_name*)', for each of the two networks created we can visualize the images (a) and (b) in (Figure 12).

We have validated and tested both ANNs to prove the performance and accuracy of the feed-forward and narx networks. Tests have been performed by calculating the MSE of the estimated values of ω_r and M_e by applying regression. The estimated values should be tested with all the data available (training, validation, and test) up to that point. If the accuracy of the results is not satisfactory, the previous steps should be repeated, as shown in the diagram in Figure 10.

3.5. Exporting the Simulink Block

At this stage, after checking the efficiency and accuracy of the ANN generated in the previous phase of the method, the neural network can be exported as a Simulink block using the command 'gensim(*net_name*)' of the DLT application. The steps to obtain an accurate estimate of the ω_r and M_e can be carried out as in the listing below. This code shows that the ff ANN has been selected to obtain these predictions and that it has two hidden layers of 10 and 5 neurons, respectively,

1. Implement a Simulink model of the IM;
2. Simulation running of the Simulink model;
3. Collect the input signals (currents, voltages, load torque, stator synchronous speed)
4. Collect the output signals, i.e., values of rotor speed and electromagnetic torque for training the ANN;
5. Design the ANN as a result of issuing similar commands as the next ones: ff_net = feedforwardnet ([20, 10]);
   ```
   ff_net.trainFcn = 'trainlm';
   ff_net.trainFcn = 'trainlm';
   ff_net.divideFcn ='dividetrain';
   ff_net.divideMode ='sample';
   ff_net.trainParam.Epochs = 1000;
   ```
6. Configure and train the ANN as a result of the commands:
   ```
   ff_net = configure(ff_net, input_signals, output_signals); [ff_net,tr] = train(ff_net,
   input_signals, output_signals).
   ```

3.6. ANN Parallelization Assessment

A fully connected network that contains two hidden layers of size 2K has been defined, and one of them is constituted as a 50% dropout layer between them to avoid overfitting. Each of the K-sub-networks is susceptible to be run in parallel. The optimal value of K will depend on the number of CPU/GPU cores available on the machine and all K models are started in parallel to follow a process that interchanges training and model validation by coding the algorithm to operate on separate data partitions, according to the SPMD parallel data execution paradigm. The assessment, which is carried out during the execution of the application, consists of the comparison and fitting of the hyperparameters of the cyber-physical model by using for this purpose the Simulink model of the IM drive, see (Figure 7). In this way, it is possible to accelerate the convergence of the optimal values with respect to the application of the approximate gradient descent only.

4. ANN-Based Speed Estimator

The input and output signals of ω_r and M_e estimators can be obtained during the training phase by simulation of the IM drive, shown in (Figures 7 and 8). ff ANN rotor speed estimator is exported as a Simulink block with the input *x1* (*time*, *currents*, *voltages*) and the output *y1* (*rotor_speed*) signals, as they are shown in (Figure 13).

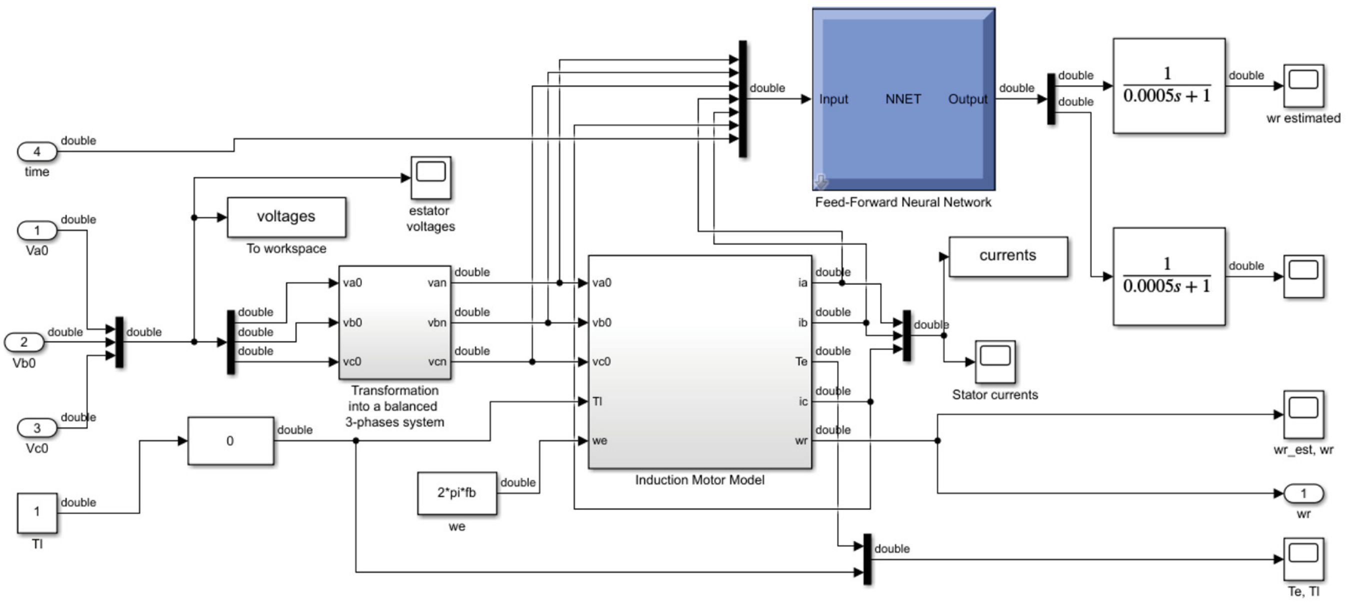
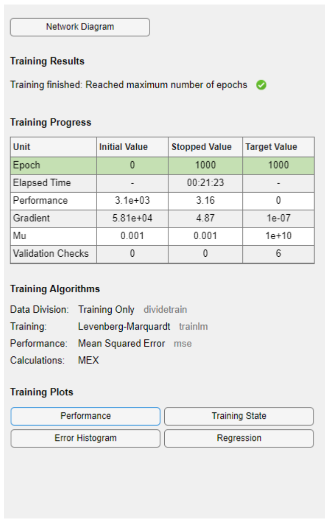


Figure 13. The construction of the *ff* ANN block in Simulink. Detailed implementation of the IM drive Simulink model, including the generated block.

4.1. FF Network Based Rotor Speed Estimator

The MATLAB DLT application has been used to train, validate, and test the ANN (created as 'ff_net'). Figure 14 shows the architecture of the ff ANN, information about the algorithm selected for training and the objective function deployed in the validation phase, as well as other characteristics of 'ff_net' and the total execution time spent training this network.



Training Results

Training finished: Reached maximum number of epochs ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:21:23	-
Performance	3.1e+03	3.16	0
Gradient	5.81e+04	4.87	1e-07
Mu	0.001	0.001	1e+10
Validation Checks	0	0	6

Training Algorithms

Data Division:	Training Only	dividetrain
Training:	Levenberg-Marquardt	trainlm
Performance:	Mean Squared Error	mse
Calculations:	MEX	

Training Plots

(**a**)

Training progress			
Unit	**Initial value**	**Stopped value**	**Target value**
Epoch	0	1000	1000
Elapsed time	–	00:21:23	–
Perfor-mance	3.1×10^2	3.16	0
Gradient	5.81×10^4	4.87	10^{-7}
Mu	0.001	0.001	10^{10}
Validation Checks	0	0	6

(**b**)

Figure 14. (**a**) Architecture of the ANN for rotor speed estimator training; (**b**) results of the ANN training.

The performance of the ANN-based rotor speed ω_r estimator of the IM has been obtained by the MSE calculated during 1000 epochs of training, and it is shown in Figure 15a. The error histogram is shown in Figure 15b.

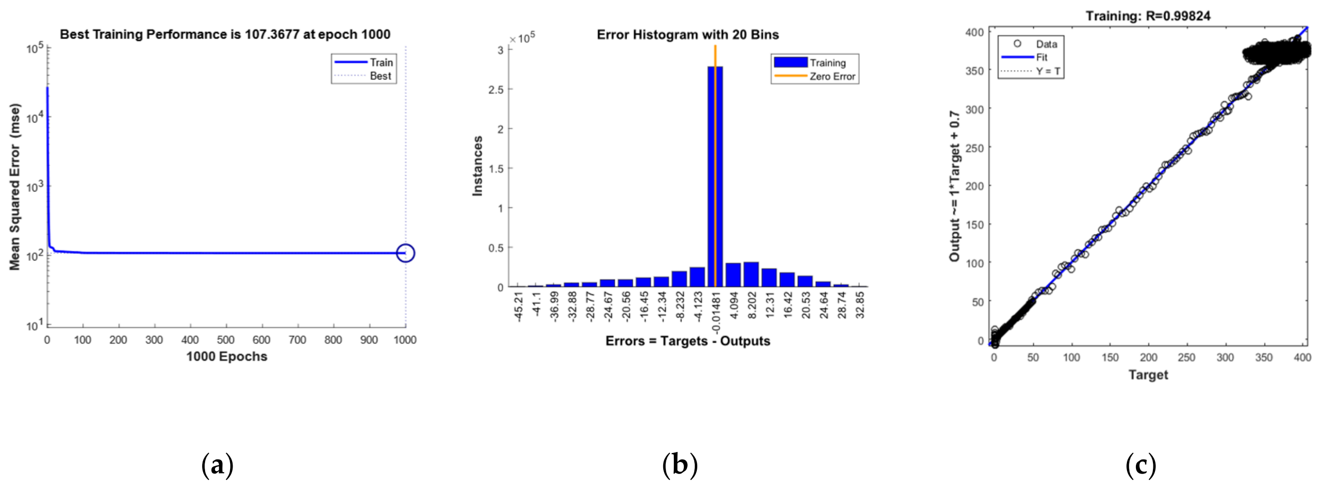


Figure 15. (**a**) Performance of the ANN-based rotor speed estimator considering mean squared error (MSE); (**b**) Error histogram of the ANN-based rotor speed; (**c**) Regression between the rotor speed estimated by the *ff* ANN and the measured in the IM drive.

As can be seen, the figures show a linear regression between the estimated (predicted by ‘ff_net’) and the measured (in the IM drive) rotor speed values. The regression calculation between the estimated and measured rotor speed ω_r has been displayed in Figure 15c.

4.2. NARX Network Based Rotor Speed Estimator

In addition to the above speed estimator, obtained from an ff ANN, another narx ANN has also been trained to predict a time series *y(t)* from the past values of the feedback signal *y(t)* and the values of the exogenous input series *x(t)*. The time series *y(t)* contains values of ω_r, and the series *x(t)*, the vectors of the motor input currents and voltages in the ‘abc’ phase-normal reference system, as Table 3 shows.

Table 3. Created narx ANN characteristics.

Series Role	Time Steps	Number of Features	Features
Predictors: *x(t)*	250,004	7	(time, i_a, i_b, i_c, v_a, v_b, v_c)
Responses: *y(t)*	250,004	2 [1]	(time, ω_r)

[1] Exogeneous signal components.

Narx neural networks can be applied in three different forms for making predictions,

- Open loop;
- Closed loop;
- Open/closed multistep prediction.

In general, with the next value of the dependent output signal *y(t)*, a regression is performed on previous values of this output signal and the previous values of the exogenous independent input signal *x(t)*.

A series-parallel architecture based on an ff network can be efficiently used for training a narx neural network when modeling dynamic systems since doing so is faster than training the parallel configuration directly. We are going to apply the created narx for an open/closed multi-step prediction, i.e., the so-called ‘narx_net’ can initially be implemented using an *ff* ANN to approximate the following function,

$$(t) = f\left(y(t-1),\ y(t-2),\ \ldots, y\left(t-n_y\right),\ x(t-1), \qquad x(t-2), \ldots\ x(t-n_x)\right) \tag{20}$$

Figure 16a shows the series-parallel architecture that has been used for training the ‘narx_net.’ Subsequently, the feedback loop is closed to convert such architecture into its parallel configuration as shown in Figure 16b for performing predictions of values in the series *y(t)*. This technique is very useful for performing multi-step forward prediction in modeling nonlinear dynamic systems such as the IM drive we are concerned with in this study.

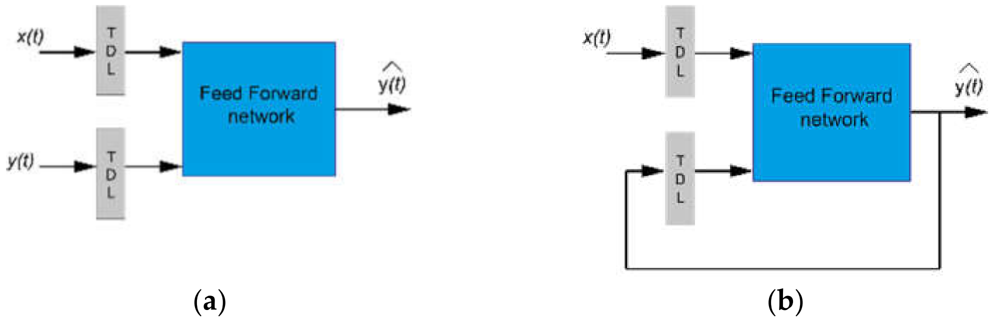


Figure 16. (**a**) Narx model implemented with feed-forward neural network to approximate the function *f*. (**b**) Parallel configuration for multi-step-ahead prediction.

The protocol to be followed to obtain an accurate estimate of ω_r and M_e is given in the following list of actions and commands of the DLT. The following code shows that the 'narx_net' has been selected to obtain the above predictions (estimates) and that it has one hidden layer of 10 neurons,

1. Implement a Simulink model of the IM;
2. Simulation running of the Simulink model;
3. Collect the input signals (currents, voltages, load torque, stator synchronous speed)
4. Collect the output signals, i.e., values of rotor velocity and electric torque for training the ANN;
5. Design the NARX ANN as a result of issuing similar commands as the next ones:
   delay1 = [1:2]; delay2 = [1:2];
   narx_net = narxnet(delay1,delay2,10); narx_net.trainFcn = 'trainlm';
   narx_net.trainParam.min_grad = 1 $\times$ 10^{-10}; narx_net.trainParam.Epochs = 1000;
6. Configure and train the NARX ANN as a result of the commands:
   [p,Pi,Ai,t]= preparets(narx_net,X,{},Y);
   narx_net=train(narx_net,p,t,Pi);
7. Close narx_net for obtaining multi-step predictions of target values:
   narx_net_closed= closeloop(narx_net);

The training data must be used with a tapped delay with respect to the inputs of both signals. The above code listing has defined two identical delays ([1:2]) for both predictors and responses so that the training of the network begins with the third data point in the *input(X)* and *output(Y)* to the series-parallel network 'narx_net.' Using the 'preparets' configuration command means that a lot of data preparation is required before training the network's tapped delay lines, which must be filled with initial conditions. Finally, the 'narx_net_closed' network is prepared to make predictions after the execution of the 'closedloop' command converts the trained network into its parallel configuration.

Since with narx neural networks, it is more frequent to get into overfitting situations, in this part of the study, it was decided to manually divide the dataset into 70% for training, 15% to validate that the ANN is generalizing correctly, and stop it before overfitting, and 15% to test the generalization performed by the ANN. Therefore, the histogram of errors and the regression between estimated data and actual data must be displayed for each of the mentioned subsets (training, validation, testing), as shown by the different graphs in Figure 17.

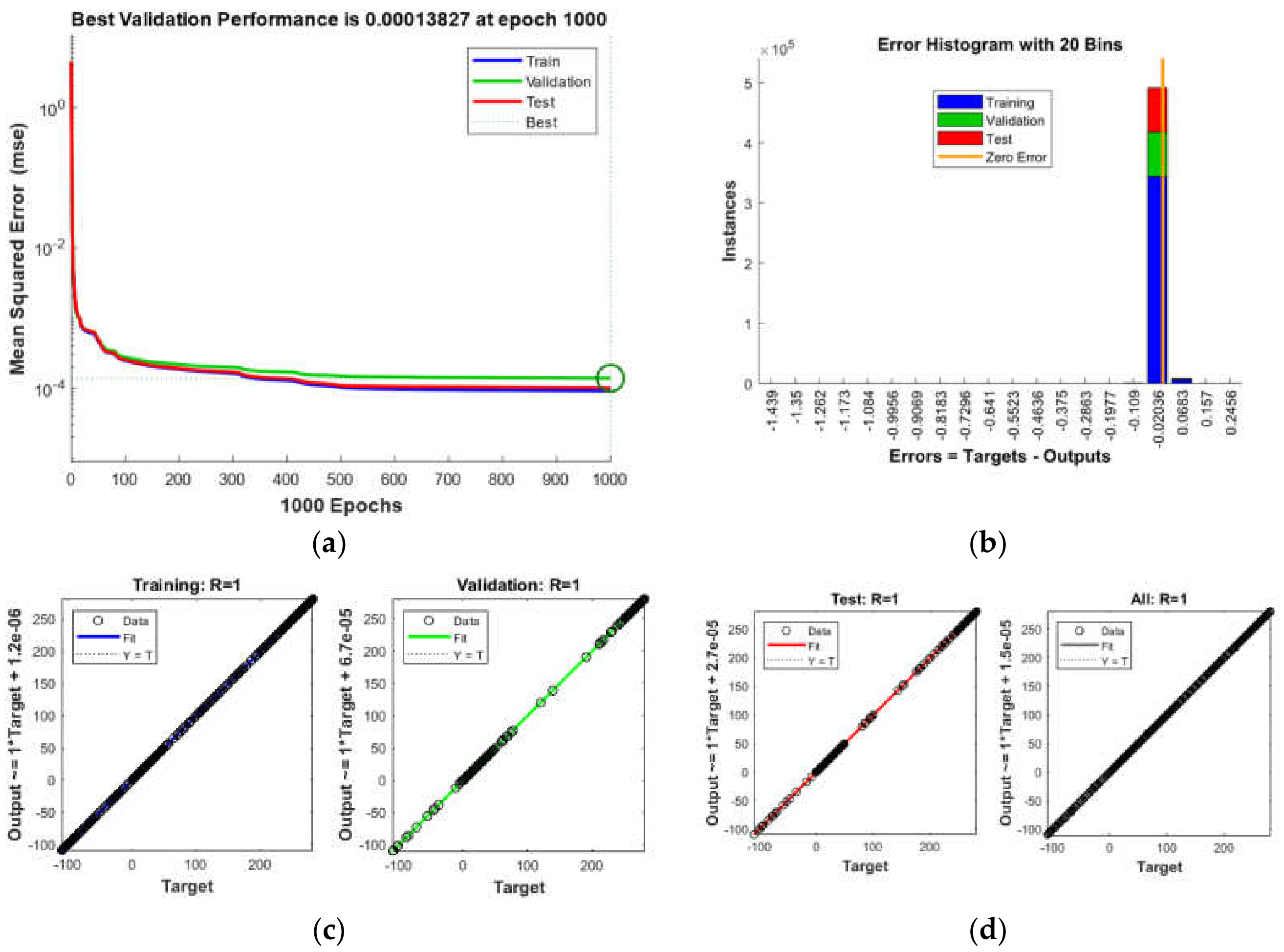


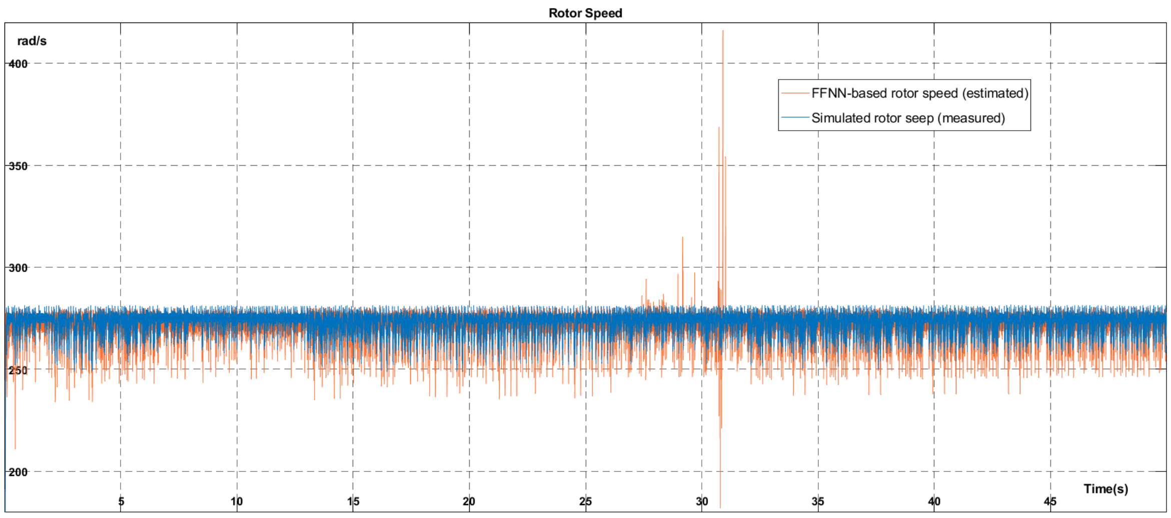
Figure 17. (**a**) Performance of the narx-based rotor speed estimator considering mean squared error (MSE); (**b**) Error histogram of the narx-based rotor speed; (**c**) Regression between the rotor speed estimated by the narx and then measured in the IM drive during training and validation; (**d**) regression during testing and regression globally calculated of the speed estimator.

4.3. Results and Discussion

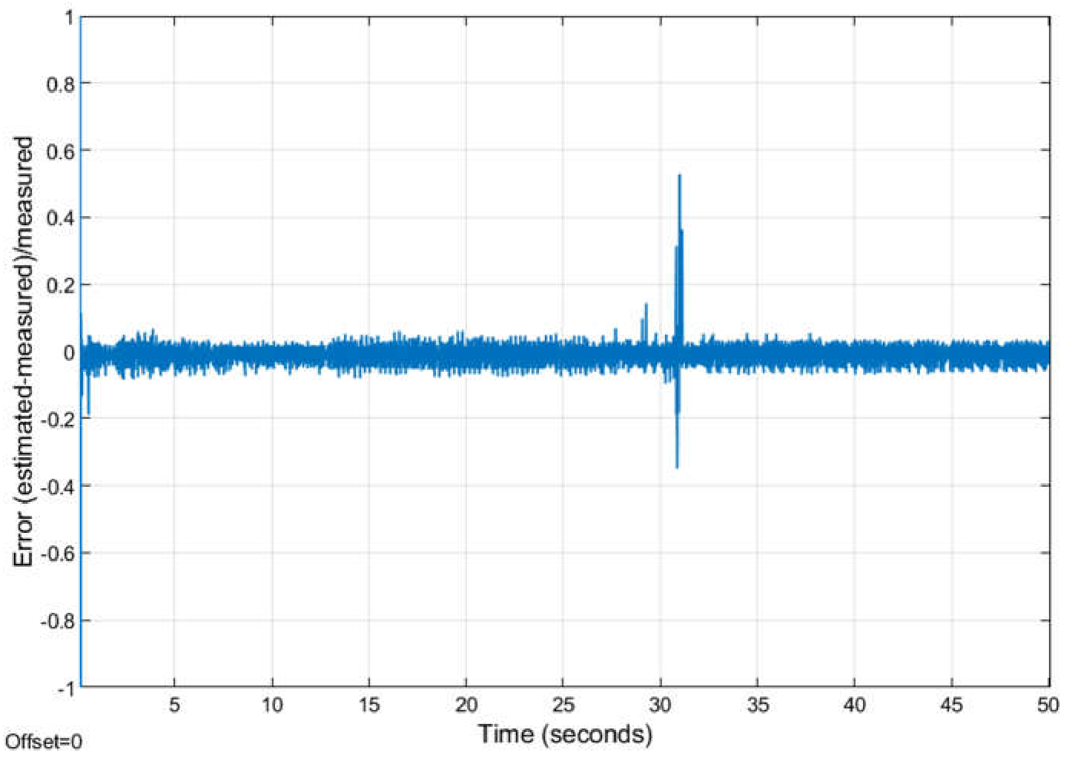
The DLT MATLAB application was used to validate the complete target system associated with the ANNs for motor speed and electromagnetic torque estimation.

We have carried out a double validation of the ANN-based estimator response for rotor speed ω_r in this section considering a time span of 50 s from the training speed and the load torque M_l has been maintained constant. The measured speed with the IM driver model and the estimated speed based on the ff ANN is shown in Figure 18. The actual reference speed of the IM driver and the speed estimated by the narx ANN-based estimator can be seen in Figure 19. We can see in both figures that the measured and estimated rotor speeds have the same trajectory, so we can conclude that the accuracy of the two-speed estimators based on two different types of ANN has been correctly validated.

However, in Figure 19, we can observe that perturbations in the response (ω_r) of the narx-based estimator are evident up to 8 s of run time, although the output signal tends to stabilize soon after. In general, we can observe that the rotor speed ω_r follows the changes produced in the synchronous ω_e angular speed in the stator. Since any change in the value of ω_e causes rapid oscillations around the new value in the rotor speed ω_r, such oscillations are more dramatic at the beginning in the narx-based estimator; this is probably caused by the closed loop of the feedback signal that brings the output values of the speed back to the input of the estimator. Therefore, we could state in view of the ω_r response plots that the narx-based estimator performs worse and takes longer to self-stabilize than the ff-based estimator. The plots in Figures 18b and 19b, which show the error between the measured and estimated values of the rotor speed, also agree that the error shown by the ff-based open-loop estimator is smaller than the error produced by the closed-loop narx-based implementation. In addition, a perturbation of the rotor speed at 31 s is shown in Figure 18a that has no effect on the estimated output speed, which confirms the self-establishing capability of the ff-based estimator.

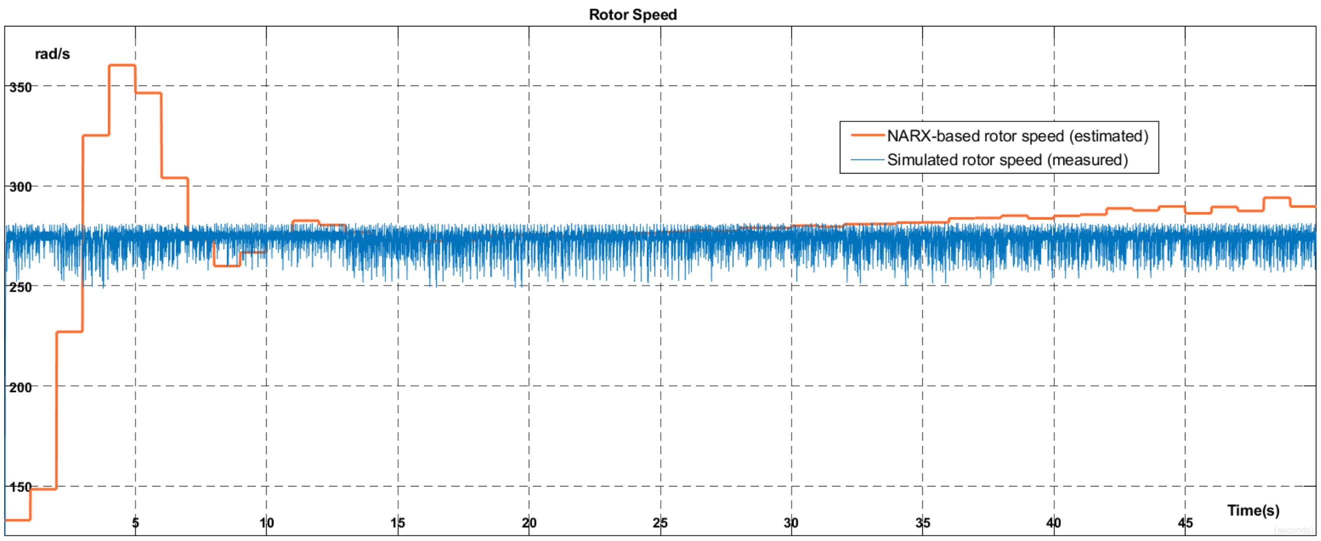


(**a**)

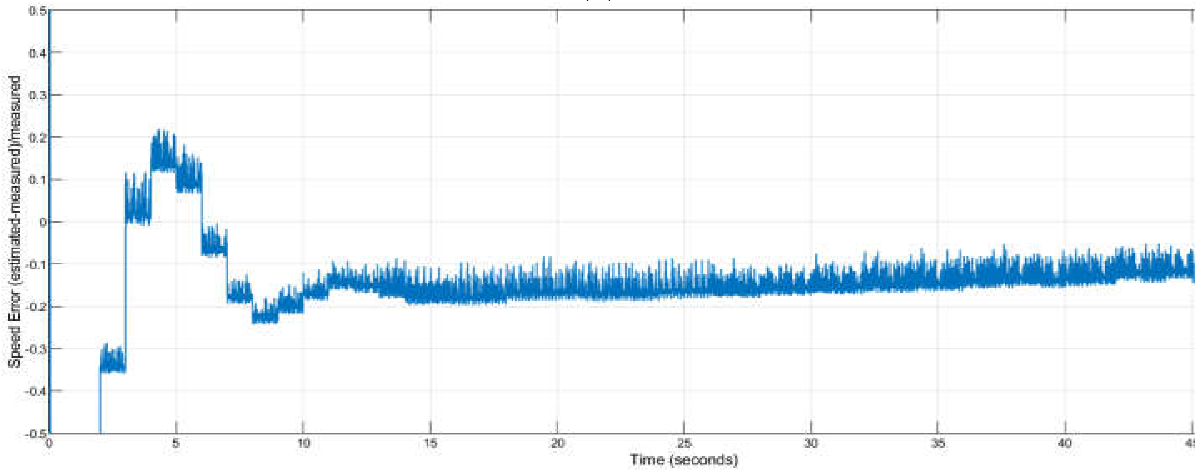


(**b**)

Figure 18. (**a**) Rotor speed ω_r measured (orange) using the IM drive Simulink model, estimated (blue) by the generated block from 'ff_net'; (**b**) Rotor speed error between measured and estimated ω_r (scaled to 1.0).



(**a**)



(**b**)

Figure 19. (**a**) Rotor speed ω_r measured (orange) using the IM drive Simulink model, estimated (blue) by the generated block from 'narx_net'; (**b**) Rotor speed error between measured and estimated ω_r (scaled to 1.0).

5. ANN-Based Electromagnetic Torque Estimator

We have carried out a double validation of the response of the ANN-based estimator for the electromagnetic torque M_e in this section, considering a time span of 50 s of the training speed ω_e, and the load torque M_l has been maintained constant.

Feed-forward and narx ANN rotor speed estimators are exported as DLT-generated blocks for their use in the target Simulink model with the input *x1* (time, currents, voltages) and the output *y1* (electromagnetic_torque) signals. The load torque M_l, which has been applied to obtain the training data, and the electrical torque M_e measured with the IM drive have been shown in Figure 20.

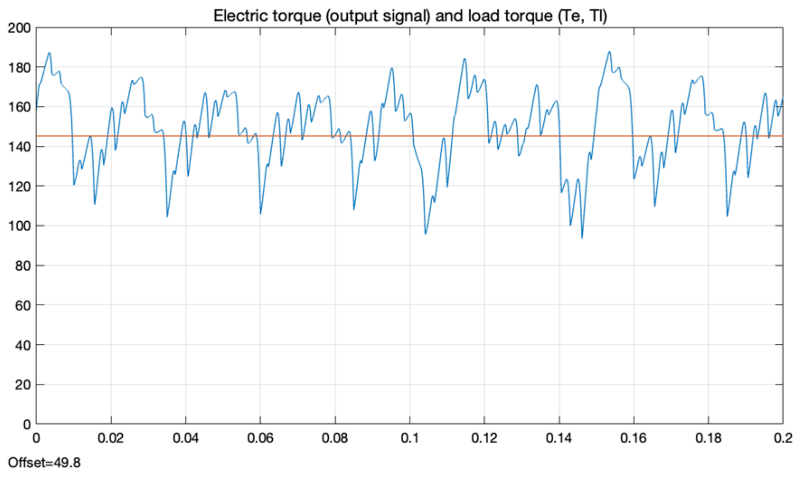


Figure 20. Zoomed electromagnetic torque (blue) M_e (Nm) with the Simulink model of the IM drive and load torque (orange) M_l (Nm) input to models during simulation.

5.1. FF Network Based Electromagnetic Torque Estimator

The input data to the ff ANN estimator are based on the measured currents and voltages in the IM drive model with respect to the *abc* phase-standard reference system. The output data are the electromagnetic torque M_e and the speed ω_r in the rotor over time.

The DLT of MATLAB has been used to train, validate, and test the ff ANN M_e estimator. The input and output data have been collected during the training by simulation with the Simulink IM drive system. The generation of the ff ANN block in Simulink is performed with the DLT commands,

1. TqTT = timeseries2timetable(Torque);
2. TqT1 = [seconds(TqTT.Time), TqTT.Data];
3. Tq = tonndata(TqT1,false,false);
4. W = [XT1(:,1:4),ZT1(:,2:4)];%(time, currents, voltages)
5. [ff_net, tr] = train(ff_net, W, Tq);
6. gensim(ff_net); %Simulink- FF ANN block generated

The performance of the ff ANN-based electromagnetic torque M_e estimator has been obtained by calculating the MSE for 1000 epochs of training, as shown in Figure 21a. The error histogram in Figure 21b shows that more than 99% of all the instances exhibit almost negligible errors. (<0.020).

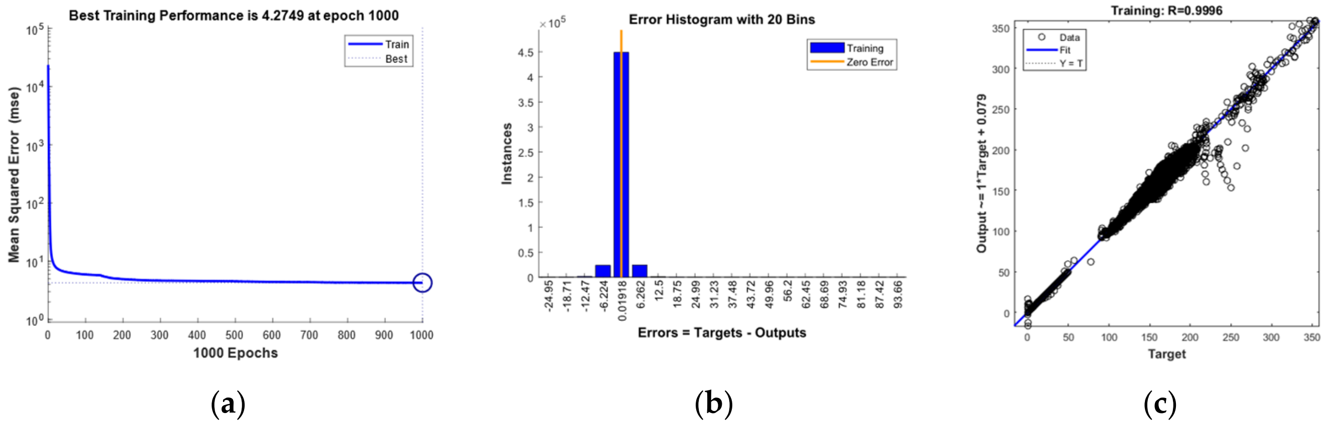


Figure 21. (**a**) Performance of the ANN-based electromagnetic torque M_e estimator considering mean squared error (MSE); (**b**) Error histogram of electromagnetic torque M_e; (**c**) Regression between the electromagnetic M_e torque estimated by the ff ANN and the measured one in the IM drive.

It can be seen that there is a linear regression between the estimated electromagnetic torque and the one measured with the IM driver, such regression has been calculated directly from the 50 s time series obtained from the Simulink model, and the results are presented in Figure 21c.

5.2. FF Network Based Electromagnetic Torque Estimator

A second narx ANN-based (Table 4) estimator of electromagnetic torque M_e for the rotor has been generated with the DLT commands,

Table 4. Created narx characteristics.

Series role	**Time Steps**	**Number of Features**	**Features**
Predictors: $x(t)$	250,004	7	(time, i_a, i_b, i_c, v_a, v_b, v_c)
Responses: $y(t)$	250,004	2 [1]	(time, M_e)

[1] Exogeneous signal components.

1. narx_net = narxnet(narx_net, delay1, delay2,10);
2. [x,xi,ai,t] = preparets(narx_net, W,{}, Tq);
3. narx_net = train(narx_net, x, t, xi, ai);
4. narx_net_closed = closeloop(narx_net);
5. %Simulink-NARX ANN block generated
6. gensim(narx_net);

The time series $y(t)$ contains now values of M_e and the series $x(t)$ the vectors of the motor input currents and voltages in the*abc* phase-normal reference system.

As we did for the speed estimator, the dataset has been manually divided into 70% for training, 15% for validation, and 15% for testing. Therefore, the histogram of errors and the regression between estimated data and actual data must be displayed for each of the mentioned subsets (training, validation, testing), as shown by the different graphs in Figure 22a–c.

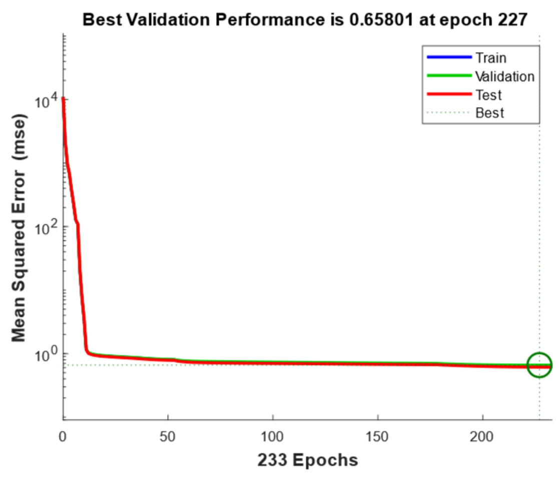
5.3. Obtained Results and Discussion

The DLT of MATLAB has been used to train, validate, and test the ff ANN and narx electromagnetic torque estimator considering a time span of 50 s from the training speed. The load torque has been maintained constant and equal to 145 Nm in the plots.

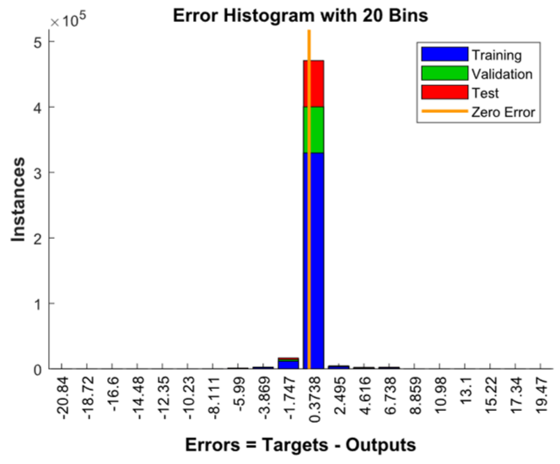
Figure 23a shows the zoomed capture of the measured data (used for training) and the estimated electromagnetic torque values after training the ff ANN estimator. The input data of the ff ANN are based on the measured voltages and currents in the *abc*-phases representation and the electromagnetic torque of the rotor. The output and input data can be obtained during the training stage by simulation of the IM drive with Simulink.

The errors, scaled to 1.0, obtained between the estimated and measured M_e for the rotor are shown in Figure 23b and the results shown validate accuracy and effectiveness of the ff ANN estimator implemented as a Simulink block by using DLT.

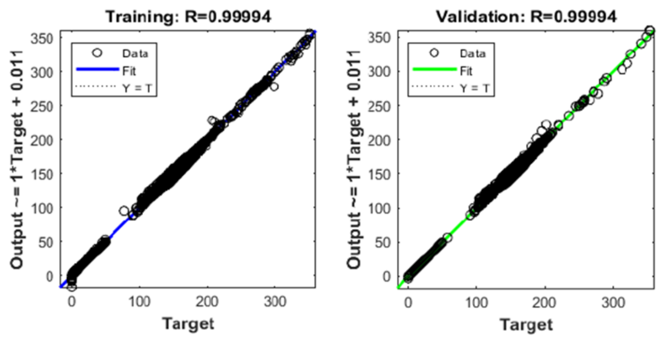
Figure 24a shows the actual reference torque M_e produced by the IM drive and the output of the narx ANN-based estimator. We can see that the measured and estimated electromagnetic torques have the same trajectory, so we can conclude that the accuracy of the two torque estimators based on two different types of ANN has been correctly validated.



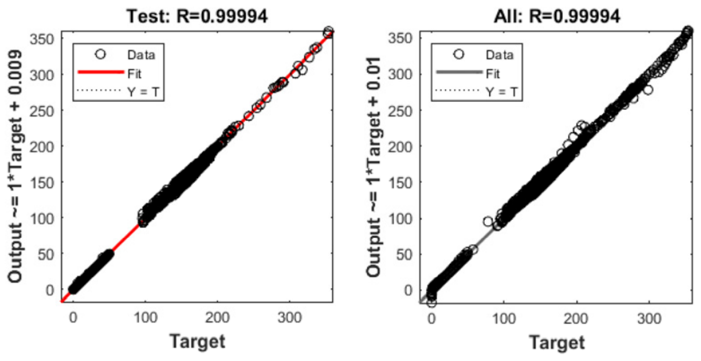
(**a**)



(**b**)



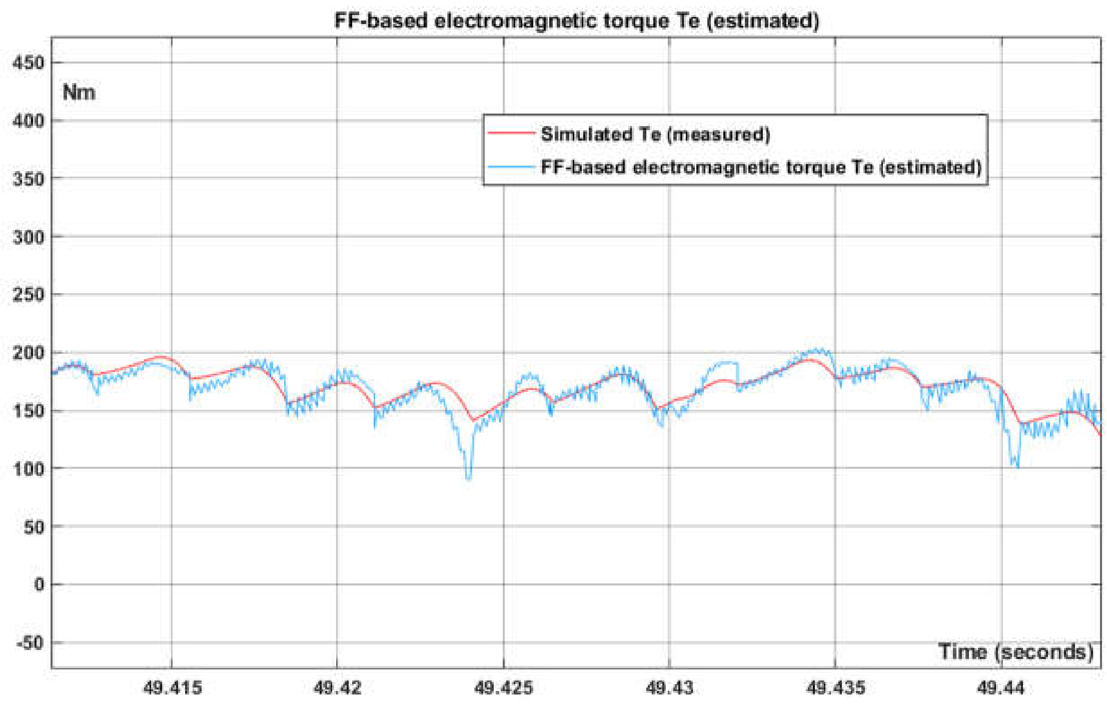
(**c**)



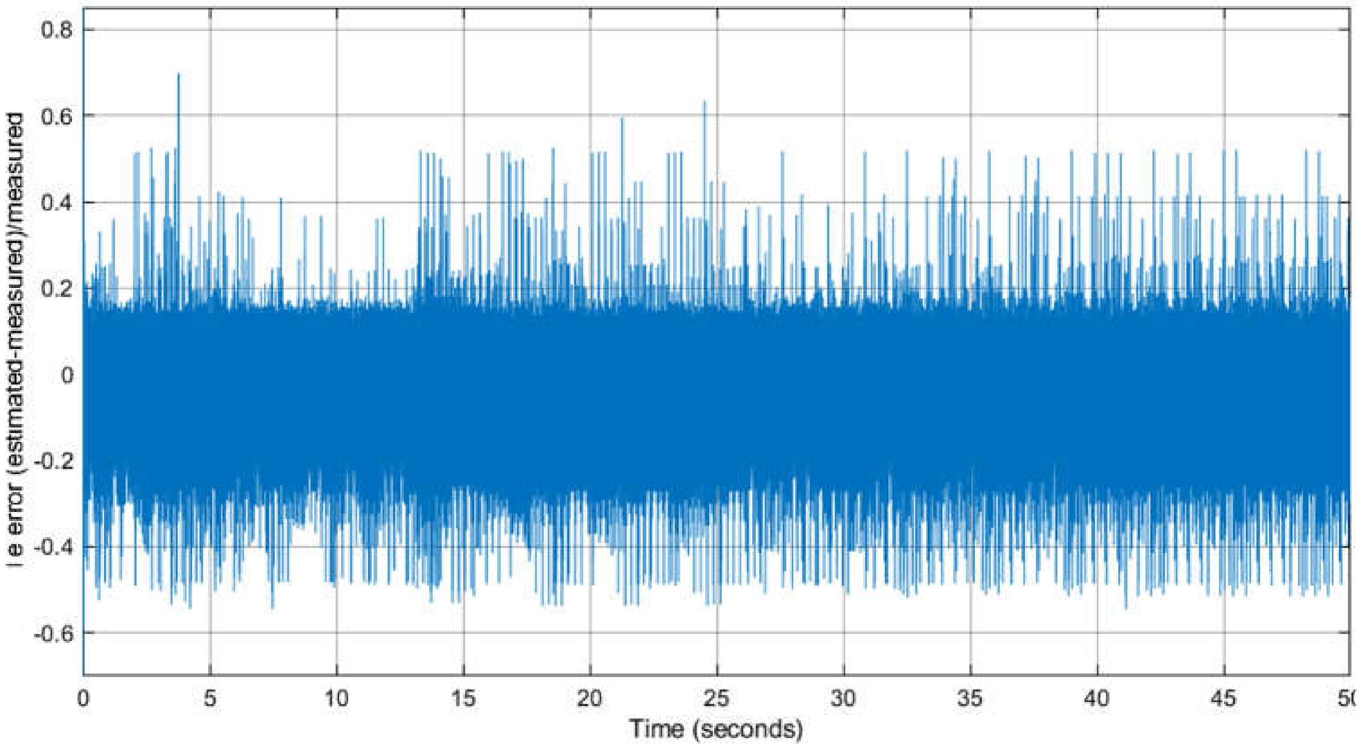
(**d**)

Figure 22. (**a**) Performance of the narx-based electromagnetic torque M_e estimator considering mean squared error (MSE); (**b**) Error histogram of electromagnetic torque M_e; (**c**) Regression between the narx-estimated electromagnetic M_e torque and the measured one in the Simulink IM drive during training andvalidation; (**d**) Regression during testing and regression globally calculated.

On the other hand, if we perform a closed loop-based estimation with the electromagnetic torque M_e controlled by the narx, exported as a Simulink block, we are apparently obtaining worse results than with the ff ANN estimator, as can be seen in the error plot between the measured and estimated values in Figure 24b. However, we can observe that the error plot of the electromagnetic torque M_e produced by the induction motor with respect to a constant load torque M_l of 145 Nm will only have significant oscillations at the beginning of the simulation, i.e., while the system is trying to reach a stabilization point. The oscillations shown represent approximately 50% of the target torque value M_e (200 Nm.), but these oscillations are caused by dynamic conditions during motor operation, which subsequently attenuates and is probably an unwanted side effect of the closed loop of the feedback signal. The initial output values of the electromagnetic torque for the rotor are not accurate. However, the estimator input is also fed with these values.

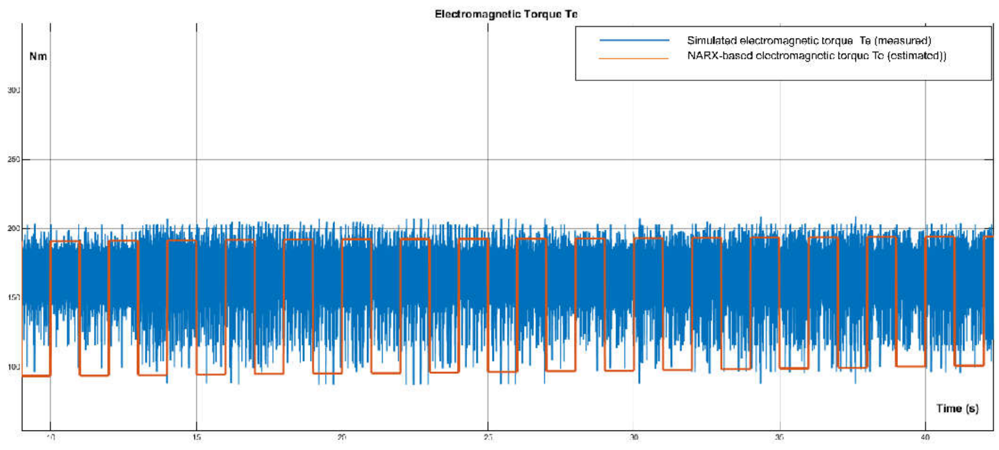


(**a**)

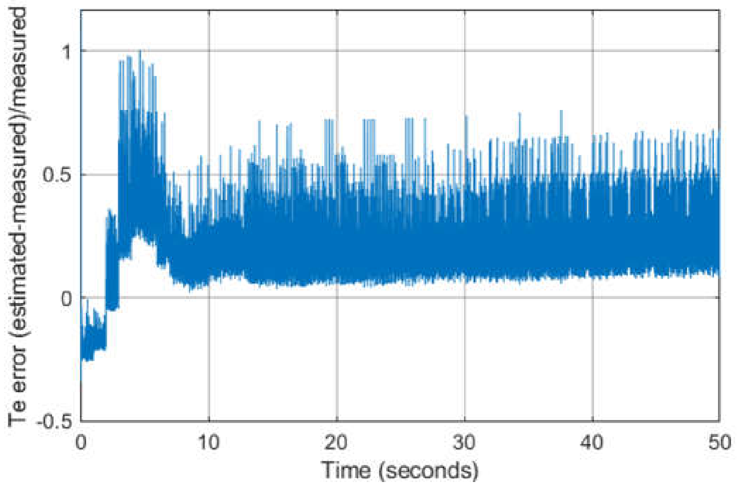


(**b**)

Figure 23. (**a**) Electromagnetic torque (orange) M_e measured for training. M_e-estimated (blue) by the ff ANN-based predictor; (**b**) M_e error (scaled to 1.0) between measured and estimated electromagnetic torque for the rotor.



(**a**)



(**b**)

Figure 24. (**a**) Electromagnetic torque (orange) M_e, measured for training. M_e estimated (blue) by the narx-based predictor; (**b**) M_e error (scaled to 1.0) between measured and estimated electromagnetic torque for the rotor.

6. Conclusions and Future Work

We have presented one method and application derivation scheme to obtain a correct control system with real-time features. Automated Machine Learning methods, together with a certain class of ANN, will allow us to model continuous and discrete dynamic systems, such as the induction motor (IM) drive that is the case study.

Four estimators, based on a feed-forward (ff) and a narx ANNs, have been used for estimating two output signals: the rotor speed ω_r and electromagnetic torque M_e of an IM drive. Simulink blocks have been implemented and analyzed for the ANN-based estimators in this article. The validation of the estimators has been performed using MATLAB and Simulink, as well as the Deep Learning Toolbox (DLT), which provides a specific command for transforming certain types of ANNs into blocks that can be directly used in cyber-physical models designed with Simulink. The training, validation, and testing of the ff and narx-based estimators have been detailed as a useful reference for engineers and researchers. The estimated rotor speed and electromagnetic torque follow the same track as the measured ones from the IM drive.

Therefore, we have shown that PID (proportional integrative differential) controllers can be substituted by trained ANN of different classes. The ff and narx have been used here to integrate continuous components in a hybrid real/time system design without any accuracy or timeliness losses. However, unlike other proposals that attempted to overcome the same problem, our methodological scheme also includes a set of guidelines as a method, which have proved to be of use for deriving a verifiable model of a cyber-physical complex system.

A possible objection to the work carried out in this paper has to do with the basic classes of ANN deployed (ff and narx), which is due to hyperparameter optimization only performed in the case study with an approximate gradient method that may not be used with Recurrent Neural Networks (RNN), which seem to be more appropriate for obtaining accurate and performant estimators of relevant cyber-physical variables in industrial systems as the induction motor drives. In future work, other classes of ANN, such as RNN, must be used and tested, which may produce an improvement in the effectiveness and estimation accuracy with respect to the ANN deployed in this paper. The case study carried out here can be continued with the detection and isolation of incipient faults to improve the safety and reliability of the induction motor with squirrel cage winding in the future.

Finally, the proposed method has been defined for its easy integration in industrial environments for simulation (Simulink). It can also be used with standard libraries for neural networks development, such as SkLearn [14] in order to interoperate with the Python NumPy and SciPy numerical and scientific libraries, and PySpark, which has been launched to support collaboration between Apache Spark and Python, is actually a Python API for Spark. PySpark allows us to use a GPU cluster architecture and SparkGPU data transfer. In future work, we plan to develop a tool capable of automated code generation of real-time and embedded system software for several computing platforms.

Supplementary Materials: The complete IM Simulink drive model supporting material can be downloaded at: https://lsi2.ugr.es/~mcapel/miscelanea/motor/ (accessed on 20 May 2022).

Funding: This research was funded by B-TIC-42-UGR20 through regional (*Consejería de Economía, Innovación, Ciencia y Empleo, Junta de Andalucía*) and EU-FEDER funds and when applicable cofounded by the Spanish Science Ministry (*Ministerio de Ciencia e Innovación*) grant PID2020-112495RB-C21.

Data Availability Statement: The main data and physical parameters of the IM drive are in the file motordata.m, which can be downloaded at: https://lsi2.ugr.es/~mcapel/miscelanea/motor/ (accessed on 20 May 2022). For the model to work, running the m-file included in Matlab MATLAB is necessary before opening the Simulink model. All the physical constants listed in Table 2 have been defined using IS physical units in the 'motordat.m' file.

Acknowledgments: I would like to thank Juan A. Holgado from the University of Granada, who collaborated in the development of an initial prototype of the IM Simulink model used in this study.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Hutter, F.; Kotthoff, L.; Vanschoren, J. *Automated Machine Learning: Methods, Systems, Challenges*; Springer Series on Challenges in Machine Learning; Springer: Berlin/Heidelberg, Germany, 2019.
2. Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; Hutter, F. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2015; Volume 28.
3. Amrhein, M.; Krein, P.T. Dynamic Simulation for Analysis of Hybrid Electric Vehicle System and Subsystem Interactions, Including Power Electronics. *IEEE Trans. Veh. Technol.* **2005**, *54*, 825–836. [CrossRef]
4. Mehrotra, P.; Quaicoe, J.E.; Venkatesan, R. Development of an Artificial Neural Network Based Induction Motor Speed Estimator. In *PESC Record, Proceedings of the 27th Annual IEEE Power Electronics Specialists Conference, Baveno, Italy, 23–27 June 1996*; IEEE: Piscataway, NJ, USA, 1996; Volume 1, pp. 682–688. [CrossRef]
5. Lv, C.; Xing, Y.; Zhang, J.; Na, X.; Li, Y.; Liu, T.; Cao, D.; Wang, F.-Y. Levenberg-Marquardt Backpropagation Training of Multilayer Neural Networks for State Estimation of A Safety Critical Cyber-Physical System. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3436–3446. [CrossRef]
6. Patel, S.A. Developing Smart Devices with Automated Machine Learning Approach: A Review. *Mater. Today Proc.* **2022**, *51*, 816–831. [CrossRef]
7. Iruela, J.R.S.; Ruiz, L.G.B.; Capel, M.I.; Pegalajar, M.C. A TensorFlow Approach to Data Analysis for Time Series Forecasting in the Energy-Efficiency Realm. *Energies* **2021**, *14*, 4038. [CrossRef]

8. Franceschi, L.; Donini, M.; Frasconi, P.; Pontil, M. Forward and Reverse Gradient-Based Hyperparameter Optimization. In Proceedings of the 34th International Conference on Machine Learning—ICML'17, Sydney, NSW, Australia, 6–11 August 2017; JMLR.org: Sydney, NSW, Australia, 2017; Volume 70, pp. 1165–1173.
9. Pedregosa, F. Hyperparameter Optimization with Approximate Gradient. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 737–746.
10. Baydin, A.G.; Cornish, R.; Rubio, D.M.; Schmidt, M.; Wood, F. Online Learning Rate Adaptation with Hypergradient Descent. *arXiv* **2022**, arXiv:1703.04782.
11. Pravin, P.S.; Tan, J.Z.M.; Yap, K.S.; Wu, Z. Hyperparameter Optimization Strategies for Machine Learning-Based Stochastic Energy Efficient Scheduling in Cyber-Physical Production Systems. *Digit. Chem. Eng.* **2022**, *4*, 100047. [CrossRef]
12. Fukuda, T.; Shibata, T. Theory and Applications of Neural Networks for Industrial Control Systems. *IEEE Trans. Ind. Electron.* **1992**, *39*, 472–489. [CrossRef]
13. Jie, R.; Gao, Y.; Vasnev, A.; Tran, M.-N. Adaptive Hierarchical Hyper-gradient Descent. *Int. J. Mach. Learn. Cybern.* **2022**, 1–22. [CrossRef]
14. Lévesque, J.-C. Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces. Ph.D. Thesis, Université Laval, Quebec, QC, Canada, 2018.
15. An, L.; Yang, G.-H. Distributed Optimal Coordination for Heterogeneous Linear Multi-Agent Systems. *IEEE Trans. Automat. Contr.* **2021**, *67*, 460–467. [CrossRef]
16. Nguyen, T.D.; Gupta, S.; Rana, S.; Venkatesh, S. Stable Bayesian Optimization. *Int. J. Data Sci. Anal.* **2018**, *6*, 327–339. [CrossRef]
17. Elsken, T.; Metzen, J.H.; Hutter, F. Neural Architecture Search: A Survey. *arXiv* **2018**, arXiv:1808.05377. [CrossRef]
18. Cheon, K.; Kim, J.; Hamadache, M.; Lee, D. On Replacing PID Controller with Deep Learning Controller for DC Motor System. *J. Autom. Control Eng.* **2015**, *3*, 452–456. [CrossRef]
19. Yuan, X.; Wang, Y. Neural Networks Based Self-Learning PID Control of Electronic Throttle. *Nonlinear Dyn.* **2009**, *55*, 385–393. [CrossRef]
20. Splater, S.A. Power Consumption Analysis of a Practical Series Hybrid Electric Vehicle. Master's Thesis, University of Illinois, Chicago, IL, USA, 1996.
21. Wu, Y.; Jiang, B.; Wang, Y. Incipient Winding Fault Detection and Diagnosis for Squirrel-Cage Induction Motors Equipped on CRH Trains. *ISA Trans.* **2020**, *99*, 488–495. [CrossRef] [PubMed]
22. Brusaferri, A.; Matteucci, M.; Portolani, P.; Spinelli, S.; Vitali, A. Hybrid System Identification Using a Mixture of NARX Experts with LASSO-Based Feature Selection. In Proceedings of the 2020 7th International Conference on Control, Decision and Information Technologies (CoDIT), Prague, Czech Republic, 29 June–2 July 2020; IEEE: Piscataway, NJ, USA, 2020; Volume 1, pp. 545–550.
23. Messai, N.; Riera, B.; Zaytoon, J. Identification of a Class of Hybrid Dynamic Systems with Feed-Forward Neural Networks: About the Validity of the Global Model. *Nonlinear Anal. Hybrid Syst.* **2008**, *2*, 773–785. [CrossRef]
24. Pomerleau, F.; Colas, F.; Siegwart, R.; Magnenat, S. Comparing ICP Variants on Real-World Data Sets. *Auton. Robot.* **2013**, *34*, 133–148. [CrossRef]
25. Treml, A.E.; Flauzino, R.A.; Suetake, M.; Maciejewski, N.A.R. *Experimental Database for Detecting and Diagnosing Rotor Broken Bar in A Three-Phase Induction Motor*; IEEE: Piscataway, NJ, USA, 2020.