doi:10.20944/preprints202205.0352.v1

Article

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

Manuel I. Capel [1,*]

[1] Department of Software Engineering, 18071University of Granada, Spain; manuelcapel@ugr.es
* Correspondence: manuelcapel@ugr.es; Tel.: +34 958 24 2816.

Abstract: A correct system design can be systematically obtained from a specification model of a real-time system that integrates hybrid measurements in a realistic industrial environment, this has been carried out through complete Matlab / Simulink / Stateflow models. However, there is a widespread interest in carrying out that modeling by resorting to Machine Learning models, which can be understood as Automated Machine Learning for Real-time systems that present some degree of hybridization. An induction motor controller which must be able to maintain a constant air flow through a filter is one of these systems and it is discussed in the paper as a study case of closed-loop control system. The article discusses a practical application of ML methods that demonstrates how to replace such closed loop in industrial control systems with a Simulink block generated from neural networks to show how the proposed procedure can be applied to derive complete hybrid system designs with artificial neural networks (ANN). In the proposed ANN-based method to design a real-time hybrid system with continuous and discrete components, we use a typical design of a neural network, in which we define the usual phases: training, validation, and testing. The generated output of the model is made up of reference variables values of the cyber-physical system, which represent the functional and dynamic aspects of model. They are used to feed Simulink/Stateflow blocks in the real target system.

Keywords Quality real-time systems, Automated Machine Learning, Real-time embedded control systems, Cyber-physical systems, Neural Networks.

1. Introduction

Automated Machine Learning (AML) methods for design and implementation of real-time systems used to specify non-functional user requirements, such as time constraints between different system actions, are also starting to be applied successfully in the requirement specification, design, and implementation phases of cyber-physical systems.

IoT is highlighting the huge current interest in developing autonomous driving to organize traffic in future smart cities, and the ability to accurately control the torque of an induction motor is crucial to enable the design of high-performance motor drive systems, such as electric vehicles (EVs) [1].

In this context, AML methods are useful to be able to find a consistent implementation of EVs and other cyber-physical systems' requirements, though they present serious application problems regarding fitness calculation, overfitting, lack of scalability and a high amount of time to compute the hyperparameters on which real-time and control systems are dependable. The problem above is worsened if we choose an artificial neural network (ANN) to calculate the hyperparameters by following an optimization method as the gradient descent one. Therefore, we propose in this paper to complement the computationally expensive training phase of ANN with hyperparameter updates carried out manually with the help of tools widely used in the industry, e.g., Simulink and Stateflow. In order to achieve dependability of the target systems and quality of their software, the values of the hyperparameters can be validated following a mixed approach that is based on the interleaving of their value updates with the training of the neural network. Tough this method could be considered as very model-specific, it allows us, however, to obtain efficiently many hyperparameters of the cyber-physical model and fulfill time constraints. Hyperparameter continuous validation technique provides the basis for obtaining a great improvement over other approaches currently more referenced in the literature, for example, obtaining hyperparameters using Bayesian optimization (HPO) [2].

Our approach works better for modelling analogic systems or those that capture measured data continuously and it is inspired in some prior results obtained in solving prediction problems with ANN [3] for Energy Efficiency systems. We aim at addressing the issue arisen if discrete components are interrelated with continuous ones, in that case the most common result leads to inflexible models, with parameters related to the physics of the system that are difficult to change during the execution of simulations.

In hybrid state machines [4] the continuous behavior described by a system of differential equations associated must change as result of the occurrence of discrete events too. Our approach gives very compact and flexible specifications of complex hybrid systems, however there are very few tools that support this class of tools by now. In our case, we hypothesize that there is no major problem in building a trained ANN, which can substitute the PID 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.

The following sections are structured as described below. First, we will present some fundamental material on AML, which is the formal foundation of our method. Next, our proposal is applied, according to the approach we will detail, to solve a problem that often occurs in the manufacturing industry, to produce a closed loop feedback system with real time requirements, necessary to maintain constant the rotor speed of an induction motor driven by a TriaC device, Figure 1, as used by an induction motor to maintain a constant air flow through a filter in HVAC systems. The case study shows how the proposed method can be applied to derive a hybrid system that also contains discrete components. In the last section, conclusions and current lines of work are presented.

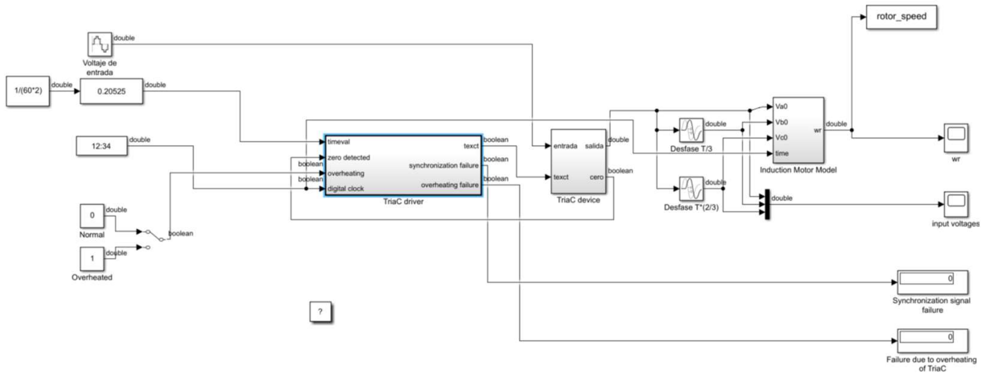


Figure 1. Simulink block diagram of an induction motor controlled by a TriaC device.

2. Automated Machine Learning

Most real-time systems problems that can be solved by algorithms or heuristics are characterized by complexities such as non-convexity, nonlinearities, discontinuities, variables of mixed nature, which involve expertise in multiple disciplines, as well as to face high dimensionality, which renders algorithms ineffective, impractical, or inapplicable in real time systems (RTS) software implementation. There are no known mathematically well-founded algorithms to find the best solution to correctly implement RTS or to solve general control problems, in general cyber-physical domains, with criticality and in a limited time.

In order to solve such problems practically, we are compelled to search new optimization algorithms, which are typically developed by using heuristics that, despite lacking strong mathematical foundations, are capable of reaching an approximate solution, in a reasonable amount of time, to the aforementioned problems. These so-called metaheuristic methods cannot guarantee that we will find the exact optimal solution we are looking for, but they can provide us with a near-optimal solution in a much more efficient way. Therefore, metaheuristic methodologies are gaining an everyday growing popularity in a variety of application domains due to their practical appeal and ease of implementation.

Most metaheuristic methods are stochastic in nature and imitate a natural, physical, or biological principle that resembles a search or optimization process. Evolutionary algorithms, more specifically, genetic algorithms and their evolution strategy; particle swarm, ant colony, bee colony optimization; simulated annealing, and a variety of other methods, are among the most used nowadays.

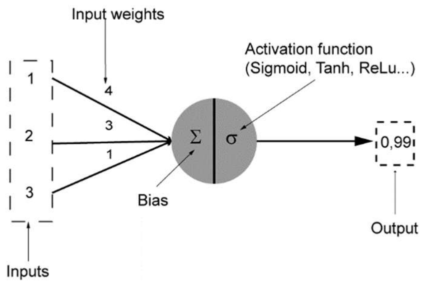
Metaheuristics have a certain advantage over traditional optimization methods, namely,

- It can provide good solutions, which may be hampered by traditional methods, for computationally easy challenges involving high input complexity;
- Can yield solutions of sufficient quality to difficult problems, e.g., problems for which no exact algorithm is known, and which can be solved in a reasonable time;
- In contrast to most conventional methods, they do not require information on gradients and can therefore be used with non-analytical black box or simulation-based objective functions;
- Most of them are inherently stochastic or deterministic heuristics that are specifically designed for this purpose and have the 'capacité' to recover from local optima;
- Since metaheuristics can better deal with inertias in goals, they are also able to recover from local optima;
- Most metaheuristics have only a few algorithmic changes to handle multiple objectives.

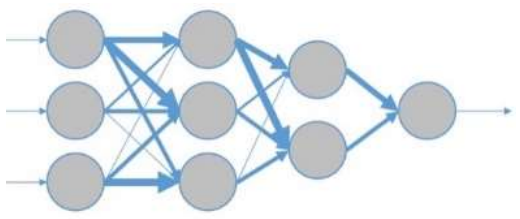
On the other hand, cyber-physical systems usually resort to excellent tools such as MATLAB/Simulink that are widely used in industry today to successfully complement, and even replace, the metaheuristic methods mentioned above in IoT or Big Data applications. Currently, the use of tools that speed up the determination of the hyperparameters of neural networks is being considered as a firm basis for the development of AML methods in an industrial environment and could certainly be considered as a sound and less complex alternative to the development of new algorithms using metaheuristics that present a high complexity both in their design and in the execution time they need.

2.1. Artificial Neural Networks and Gradient Based Optimization

Complex numerical and symbolic calculations can be made at incredible speed on modern parallel computers. However, they cannot still be close to the performance of human minds to carry out perceptual tasks such as the recognition of language and images. Computers need accurate input information and sequentially follow instruction-streams while the human mind performs tasks in a highly distributed and parallel mode. We can say, therefore, that a biological neural network is the basis for the design of an ANN and thereof it must be considered as a fundamental computation model that is intrinsically parallel and convenient to solve very complex problems in the cyber-physical domain.



(**a**)



(**b**)

Figure 2. (**a**) Artificial neuron; (**b**) Simple artificial neural network (ANN).

Just as the human brain is formed by interconnected biological neurons, an ANN is made up of artificial neurons, which are connected and grouped at different levels called layers, as Figure 2 shows. For any ANN to obtain satisfactory results, it must have been previously trained. The latter process consists of adjusting each of the weights of the inputs of all the neurons that are part of the neural network so that the responses of the output layer fit as closely as possible to the known data. The optimization algorithm normally used to train ANN is called gradient-descent. The methods based on gradient descent act as a measurement device to gauge the accuracy of the cost function calculation with each step or iteration of the ANN parameters update.

2.1.1. Optimization and gradient descent

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 tune hyperparameters of cyber-physical models [2]. Following this approach, evaluations of the target function results into a hyper-gradient vector instead of single values that are usually obtained by other methods such as the called hyperparameter optimization (HPO).

Thanks to parallel computing we can now handle many hyperparameters of a model [5] [6] by deploying gradient descent-based methods efficiently, more specifically, our method for ANN applied to high-dimensional HPO problems can perform the following tasks,

- Separately optimize the learning rate of an ANN for each iteration and layer;
- Calculate optimal weights for each layer in the ANN;
- Reduce the likelihood of overfitting by L2 regularization for each individual parameter.

We can, therefore, overcome the backpropagation through the complete training procedure of an ANN by carrying out hyperparameter updates offline and by possibly following a separate manual validation with "external" tools interleaved with the training of the network. This method is highly model-specific but, in return, it allows us to tune many hyperparameters of the cyber-physical model, which sets the ground for obtaining a great improvement with respect to HPO carried out, e.g., by Bayesian optimization.

2.2. Feature Processing Algorithms

To build arbitrary size ML pipelines our approach proposes to use more than one algorithm and dynamically add these algorithms to the parallel calculation to enlarge the search space and then to haste select the right algorithm and its hyperparameters [7][8].

2.3 Scalability

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 [9]. Our approach with respect to cope with the scalability problem is to take advantage of massive parallel computing and try to full exploit large-scale computer clusters or the thousands of SMs of GPU/CUDA multiprocessors.

2.4 Overfitting and generalization

Overfitting is an open problem when we try to apply ANN with a finite validation set and the calculation of the model's hyperparameters suffer from this drawback [10]. One possibility would be to use a different shuffling for each function we need to evaluate, and thus reduce the amount of overfitting. This approach has 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 it was noted in [11].

3. Hybrid Systems Simulation

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/ . 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 4. For the model to work, it is necessary to run the m-file included in MATLAB before opening the Simulink model. All the physical constants listed in Table I have been defined using IS physical units in the 'motordat.m' file.

3.1 Physical Modeling of an Induction Motor

The most difficult component to model is an induction motor is the IM drive itself, as specialized Simulink blocks are needed that include differentiation operators, or develop them manually, as Figure 4(b) shows. The other devices in the model that includes the IM can be modeled by means of simple switches or a combination of them.

An induction motor works according to the physical principle of mutual induction between electrical circuits traversed by a varying magnetic flux Φ. Applying Faraday's law, which is given by the following equation, we can obtain the magnetic flux through the winding of a motor,

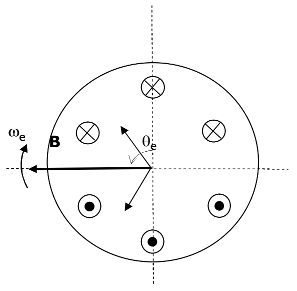
$$\varepsilon = -\frac{d}{dt}(N \cdot \Phi_B), \tag{1}$$

The magnetic flux passing through a motor winding only depends on the current conducted by the circuit. 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 being affected by magnetic induction, according to the equation,

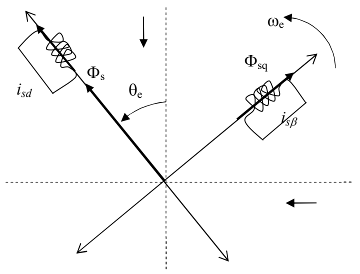
$$N \cdot \Phi = L \cdot i_{reel} \, , \tag{2}$$

The actual winding of induction motors consists of three windings, each winding drives with a voltage phase separated 2π/3 rad. from the next phase, for obtaining a rotating magnetic field in the stator core of the motor, as shown in Figure 3(a). We call synchronous speed the magnetic field's rotational speed ◎e, which is an important parameter of induction motors.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 May 2022 doi:10.20944/preprints202205.0352.v1



(a)



(b)

Figure 3. (**a**) Motor winding; (b) Rotating reference system.

If we short-circuit the rotor winding of a three-phase wound induction motor, then a current is induced which produces an electromagnetic force, and the motor will start to rotate at the speed ω_r due to the change in the direction of the magnetic field, which is a consequence of the synchronous angular speed ω_e of the magnetic field $\vec{B}$ in the stator. The difference between both rotational speeds (ωe and ωr) is another parameter of induction motors, which is called *slip*.

Table 1. Constants and variables of the physical variables of an induction motor.

Observables and references of the IM physical system	
d: direct axis of the rotating reference system	$\chi^*_{lm}= 1/(1/\chi_{ls}+1/\chi_{lr}+1/\chi_m)$: 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
$F_{ij}=\Phi_{ij}$, 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**)
χ_{ls}: stator reactance ($\omega_e \cdot L_{ls}$)	$\omega_b=2\cdot\pi\cdot f_b$: angular speed corresponding to the electric frequency of the motor feeding voltage.
χ_{lr}: rotor reactance ($\omega_e \cdot L_{lr}$)	ω_r: rotor angular speed (**output variable**)

3.2 The two phases synchronous rotating frame

We can assume a reference system that rotates at the synchronous speed ω_e of the stator to ease the representation of the rotating magnetic field $\vec{B}$ and the inductance linkages $\vec{\phi}$, as shown in Figure 3(b). The induction motor is therefore modeled by two reels, the first one intended to drive the current in the stator to generate the rotating magnetic field. The second reel will generate the magnetic field induced in the rotor and the force that makes it rotate. We can very accurately represent the magnetic coupling between the stator and rotor windings of an induction motor with this simple model. The first axis in the Figure 3(b) is called the direct axis (d) and the second one is called the quadrature axis (q), and both constitute the rotating reference system d-q.

In addition, the rotor angle θ_e must also be considered as another important parameter of the induction motor model. An induction motor with a short-circuited rotor winding (called "squirrel cage") will keep its voltages v_{qr} and v_{dr} at zero. All the values of constants

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 May 2022 doi:10.20944/preprints202205.0352.v1

and variables of the linear differential equations system of the above model have been summarized in Table 1.

3.3. Calculation of the electromagnetic torque generated by the motor

Now, applying the theory of electromagnetism, we can obtain a mathematical expression that serves to calculate the electromagnetic torque generated by the motor. We will derive, therefore, the mechanical power given by the motor from the following electromagnetic equation,

$$P_{mech} = \frac{3}{2}\left(\omega_b \cdot \phi_{sq} - \omega_b \cdot \phi_{sd} \cdot i_{sq}\right) \tag{3}$$

and the magnetic linkage $\phi_{sq,\,sd}$ only depends on the angular speed and the magnetic flux $F_{ij} = \omega_e \cdot \Phi_{ij}$. The mechanical power can be made equivalent to the electrical torque M_e generated by the motor, then we can obtain,

$$M_e = \frac{3}{2}\left(\frac{p}{2}\right)\frac{1}{\omega_b}\left(F_{ds}i_{qs} - F_{qs}i_{ds}\right) \tag{4}$$

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 4(b), with concrete values of p (the number of poles) and ω_e (stator speed) as the input data to the induction motor model.

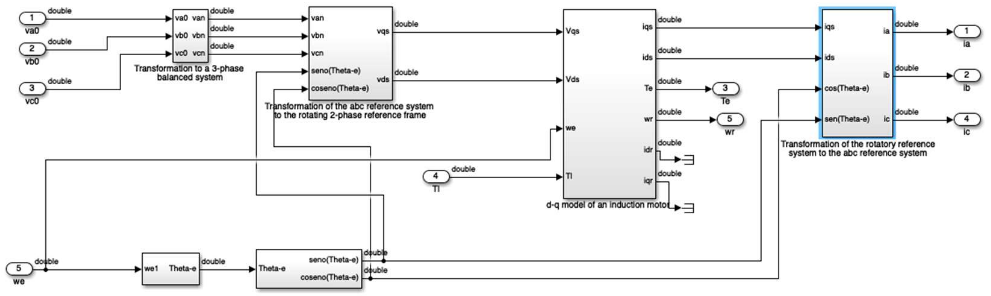
The angular speed ω_r of the rotor can also be calculated since the load torque M_l and inertia J are also parameters of the induction motor model. As the above equation (4) shows, the electrical torque and the angular rotor speed depend on the number of poles of the rotor winding, on the contrary of what happens with the magnetic couplings

3.4 Model of an Induction Motor Drive

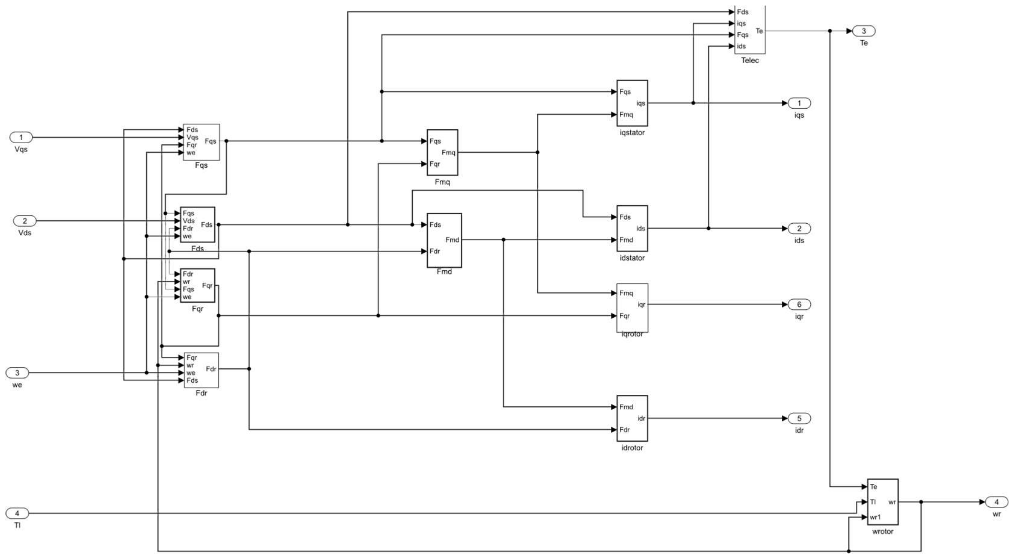
The model of an induction motor can be structured in 3 main blocks, Figure 4(a):

1. Transforms the three stator voltages v_a, v_b, v_c , with a phase of 2π/3 between each two, into the *rotating reference system d-q*;
2. The block representing the induction motor itself (which inputs the three phase voltages, the synchronous angular speed ω_e and the load torque M_l);
3. This block returns the expression of the model variables in the d-q system back to the three phases 'abc' reference system, since the latter one gives us the standard graphical representation of currents in the stator.

Two specific blocks have also been designed, one to calculate the electromagnetic torque M_e given by the motor and another to calculate the rotor angular speed ω_r, in Figure 4(b).


1
va0
2
vb0
3
vc0
double
va0 van
vb0 vbn
vc0 vcn
Transformation to a 3-phase balanced system
van
vbn
vcn
seno(Theta-e)
coseno(Theta-e)
vqs
vds
Transformation of the abc reference system to the rotating 2-phase reference frame
Vqs
Vds
we
Tl
iqs
ids
Te
wr
idr
iqr
d-q model of an induction motor
3
Te
5
wr
4
Tl
iqs
ids
cos(Theta-e)
sen(Theta-e)
ia
ib
ic
Transformation of the rotatory reference system to the abc reference system
1
ia
2
ib
4
ic
5
we
we1 Theta-e
Theta-e
seno(Theta-e)
coseno(Theta-e)

(**a**)



(**b**)

Figure 4. (**a**) Simulink model of the IM; (b) The block "*dq-model of…*" that implements the IM drive in the d-q rotating reference system.

4. Modeling Method

In the proposed method we will use different ANN classes [3] 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 5). The output generated or "measured" from the IM Simulink model, which represent the functional and dynamic aspects of model, are the training data.

During the execution of the system model implemented with Simulink, the main variables of the cyber-physical system acquire values that are used to train the ANN. The MATLAB Deep Learning tool allows us to generate Simulink blocks that implement different types of neural networks (Feed Forward, NARX, …).

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.

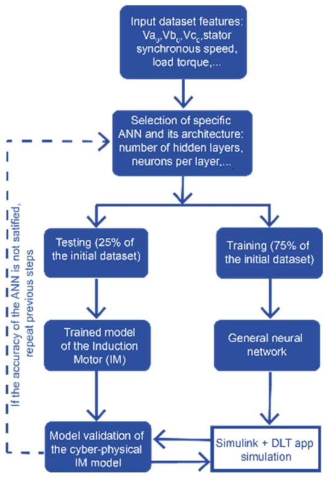

Input dataset features: Va, Vb, Vc, stator synchronous speed, load torque,…
Selection of specific ANN and its architecture: number of hidden layers, neurons per layer,…
Testing (25% of the initial dataset)
Training (75% of the initial dataset)
Trained model of the Induction Motor (IM)
General neural 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 5. 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.

4.1 Registry

The training data were obtained by simulation using the Simulink model of the IM drive shown in Figure 4. 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 [13].

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 the stator (see Table 1).

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 [14], 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, in order to accurately predict the values of the above variables, must be able to react and self-stabilize against variations of input voltages, the load torque M_l and stator angular speed w_e, to achieve this the IM drive Simulink model proves to be of great help.

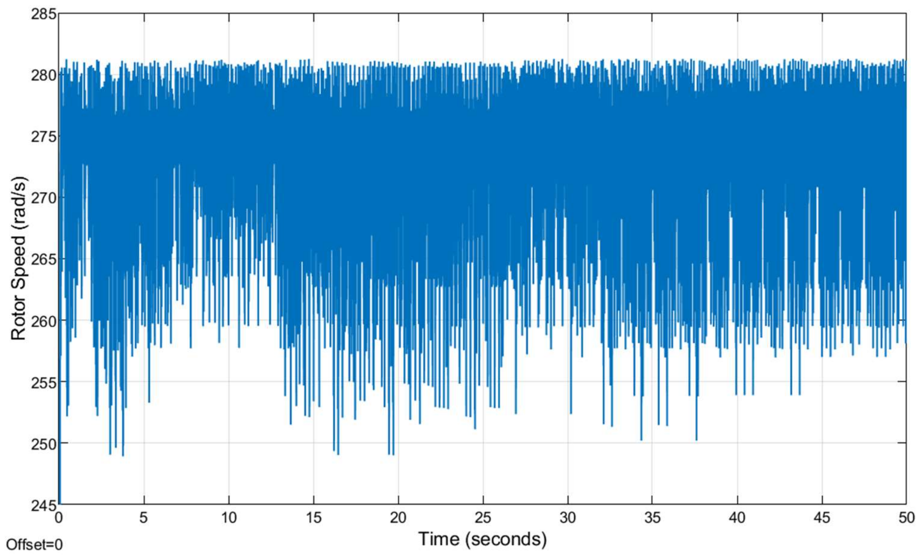


Figure 6. Applied reference rotor speed. Training data are collected from the Simulink model of the IM drive.

The training data have been collected assuming initially a reference stator speed ω_e= $2 \cdot \pi \cdot f_b$, where f_b is the frequency of the feeding voltage (=100 s^{-1}.) in the stator winding and M_l is the load torque, whose values are in the range [0.0…330.0] N×m 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 6 shows the reference rotor speed ω_r, which is assumed to change over the simulation time. The simulation time was 50 s. The simulation has been done with a sample time of 0.0005 s. and the size of data points is 250,000 points. 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.

4.2 Design of the ANN

The type of ANN is very important to get an accurate estimation of the electromagnetic torque and speed of the IM. The MATLAB Deep Learning toolbox has been used to implement the ANN deployed in this study.

A shallow multilayer neural network with 2 hidden layers has been initially defined, Figure 7 (a). 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 (50s.). We have selected here 20 and 10 neurons for the first and second hidden layers, respectively.

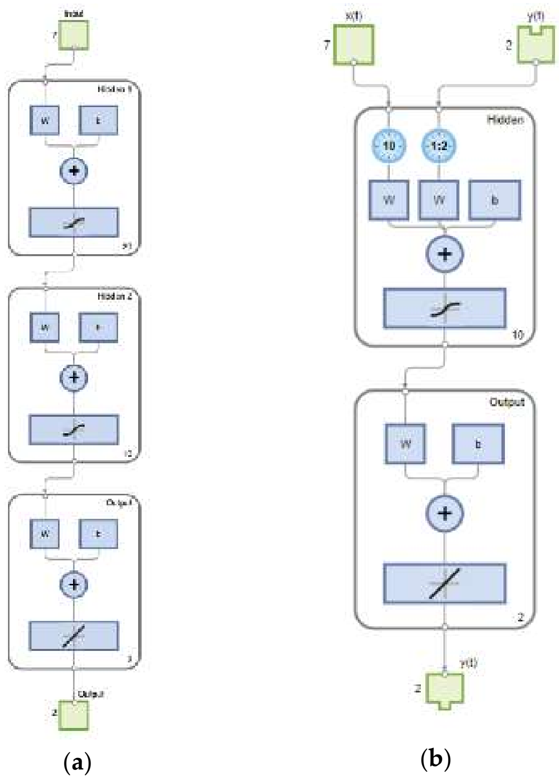


Figure 7. Architectures of the neural networks deployed in the study.

In the second part of the study a non-linear autoregressive with exogeneous input neural network has been used, Figure 7 (b). A 2-layer feed forward NARX, with 10 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, 1 linear transfer function (Simulink block with an oblique straight line) has been defined.

4.3 Training the ANN

The main objective of ANN training is to minimize the objective function iteratively by fitting $\Pi_i = \{\alpha,\beta,\gamma,\ldots\}$ 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.,

$$x = \{v_a^i, v_b^i, v_c^i, i_a^i, i_b^i, i_c^i, M_l^i, \omega_e^i\}_{i=1}^K \quad (5)$$

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

$$F(x,\Pi_i) = l.r.(\{v_a^i, v_b^i, v_c^i, i_a^i, i_b^i, i_c^i, M_l^i, \omega_e^i\}_{i=1}^K, \Pi_i) + r(\Pi_i) \quad (6)$$

Where F is the objective function, *l.r.* represents a linear relationship between each set of samples and the hyperparameters Π_i, and the function $r(\cdot)$ that 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, gradient descent, … In our method we have chosen the latter because the functions describing the cyber-physical model are all differentiable and 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 voltages data during a representative run of the IM drive model, to save implementation time. The motor speed 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.

4.4 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, which we have named MSE ("mean square error") model based on four key factors: v_x, i_x, ω_r and the M_e. (see definitions in Table 1) affecting the reliability and efficiency of the IM model.

Therefore, we can interleave the dimensions of the model ($\overrightarrow{v_x}$, $\overrightarrow{\iota_x}$, 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,

$$I_{k_x} = \frac{\sqrt{\sum_{i=1}^K \left(\overrightarrow{\iota_{x_i}} - \overrightarrow{\iota_{x_i}^{ref}}\right)^2}}{K} \quad (7)$$

$$E_{k_x} = \frac{\sqrt{\sum_{i=1}^K \left(\overrightarrow{v_{x_i}} - \overrightarrow{v_{x_i}^{ref}}\right)^2}}{K} \quad (8)$$

$$M_{e_k} = \frac{\sqrt{\sum_{i=1}^K (M_{e_i} - M_{e_i}^{ref})^2}}{K} \quad (9)$$

$$W_{r_k} = \frac{\sqrt{\sum_{i=1}^K (\omega_{r_i} - \omega_{r_i}^{ref})^2}}{K} \quad (10)$$

The reference values of the 4 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

$$F(\Pi) = \log(\alpha \cdot E_{k_x} + \beta \cdot I_{k_x} + \gamma \cdot M_{e_k} + \delta \cdot W_{r_k}) \quad (11)$$

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([20 10])` of DLT; i.e., we create a 2 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 2 networks created we can visualize the images (a) and (b) in Figure 7.

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 of Figure 5

4.5 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 electro-
    magnetic torque for training the ANN
5.  Design the ANN as result of issuing similar commands as the next
    ones: ff_net=feedforwardnet([20 10]);
     ff_net.trainFcn= 'trainlm';
     ff_net.divideFcn='dividetrain';
     ff_net.divideMode='sample';
     ff_net.trainParam.Epochs=1000;
6.  Configure and train the ANN as result of the commands:
     ff_net= configure(ff_net, input_signals, output_signals);
     [ff_net,tr]=train(ff_net, input_signals, output_signals);
```

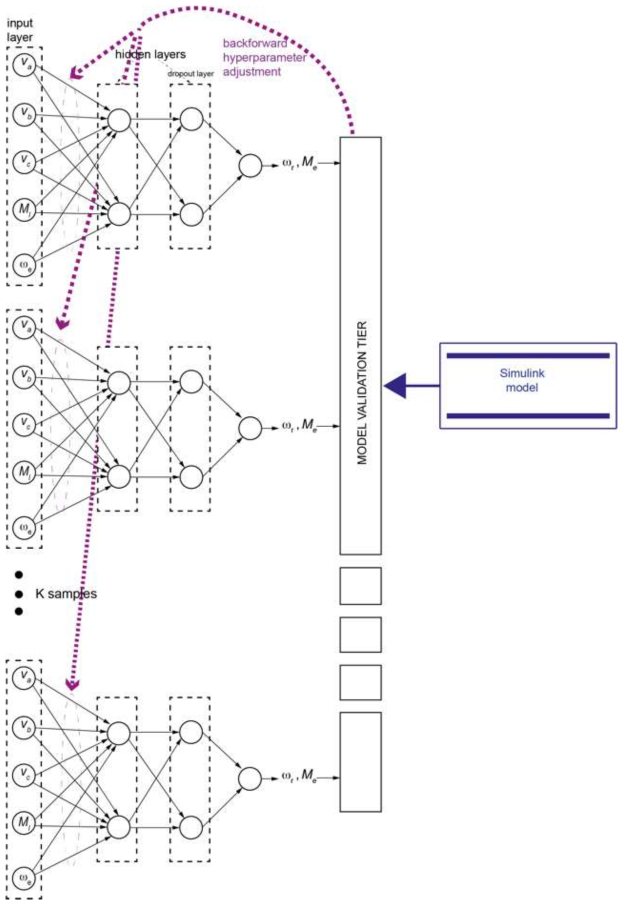


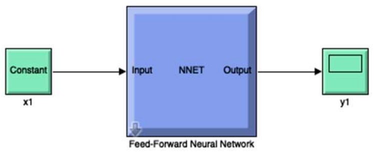
Figure 8. The parallel architecture of the generated models.

4.6 ANN Parallelization Assessment

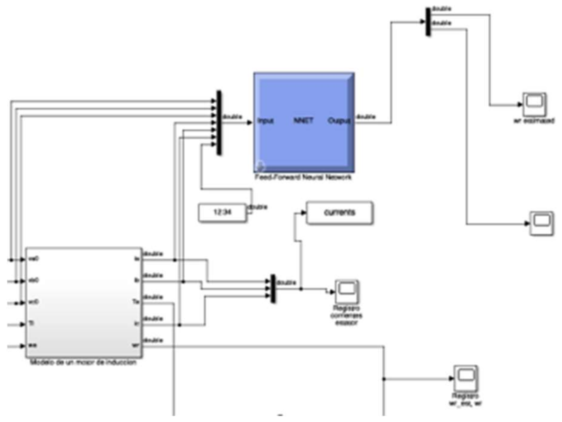
A fully connected network that contains two hidden layers of size 2K has been defined, and one of them is constituted as a 50% dropout layer between them to avoid overfitting. As shown in the parallel model of Figure 8. Each of the K-sub-networks is susceptible to be run in parallel. The optimal value 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 system by using for this purpose the Simulink model of the IM drive, see Figure 4(b). 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

5. ANN-based Speed Estimator

The input and output signals of ω and M_e estimators can be obtained during the training phase by simulation of the IM drive, which construction is shown in Figure 4. Feedforward 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 9(a).



(a)

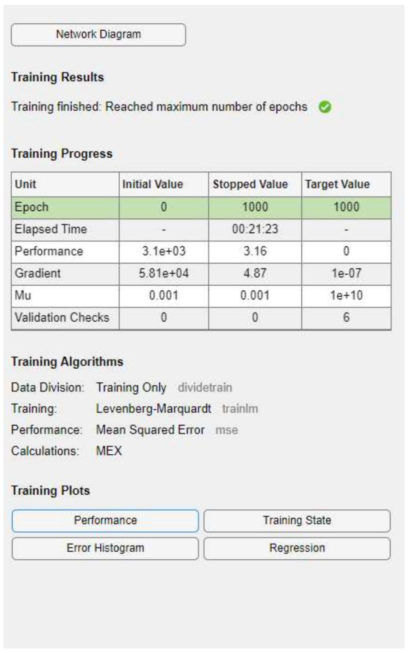


(b)

Figure 9. (**a**) The construction of the FF ANN block in Simulink; (**b**) Detailed implementation of the IM drive Simulink model including the generated block.

5.1 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 10 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.



Network Diagram

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

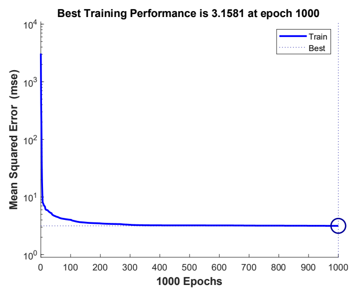
Training Algorithms

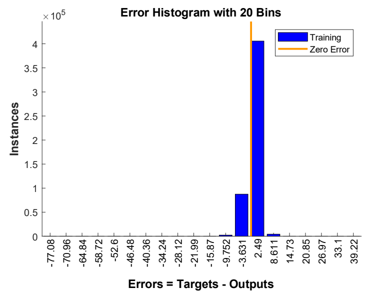
Data Division:	Training Only	dividetrain
Training:	Levenberg-Marquardt	trainlm
Performance:	Mean Squared Error	mse
Calculations:	MEX	

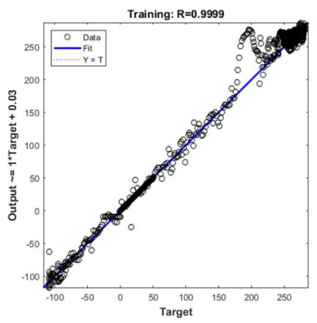
Training Plots

Figure 10. Architecture of the ANN for rotor speed estimator training.

The performance of the ANN-based rotor speed ωr estimator of the IM has been obtained by the MSE calculated during 1000 epochs of training and it is shown in Figure 11(a). The error histogram is shown Figure 11(b).







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

Figure 11. (**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 it 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 ωr has been displayed in Figure 11(c).

5.2 NARX Network Based Rotor Speed Estimator

In addition to the above speed estimator, obtained from an FFANN, another NARX 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 exogeneous 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.

Table 2. Created NARX characteristics.

Series role	Time steps	Number of features	Features
Predictors: $x(t)$	250004	7	(time, i_a, i_b, i_c, v_a, v_b, v_c)
Responses: $y(t)$	250004	2 [1]	(time, ω_r)

[1] Exogeneous signal components.

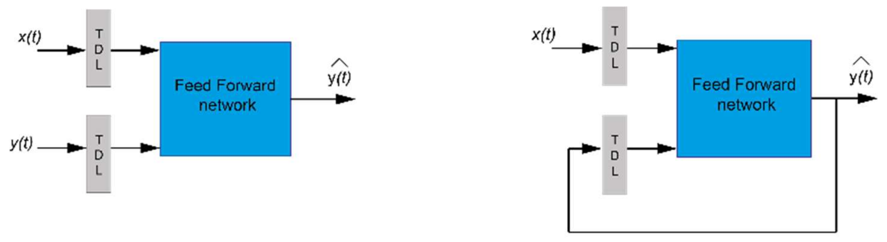
NARX neural networks can be applied in three different forms for making predictions,

- 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 ANN 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 a feed forward ANN to approximate the following function,

$$y(t) = f(y(t-1), y(t-2), \dots, y(t-n_y), x(t-1), \quad x(t-2), \dots x(t-n_x)) \tag{12}$$



(a) (b)

Figure 12. (a) NARX model implemented with feed forward neural network to approximate the function f. (b) Parallel configuration for multi-step-ahead prediction.

Figure 12(a) 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 12(b) 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.

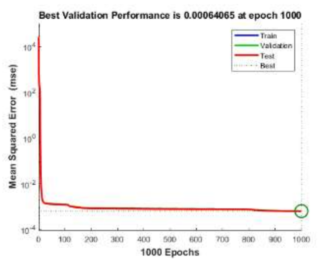
The protocol to be followed to obtain an accurate estimate of ω_r and M_e is given in the following list of actions and commands of the DLT. The following code shows that the 'narx_net' has been selected to obtain the above predictions (estimates) and that it has 1 hidden layer of 10 neurons,

```
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 elec-
    tric torque for training the ANN
5.  Design the NARX ANN as result of issuing similar commands as the next
    ones: delay1=[1:2]; delay2=[1:2];
```

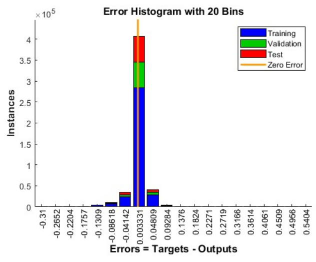
```
   narx_net=narxnet(delay1,delay2,10); narx_net.trainFcn= 'trainlm';
   narx_net.trainParam.min_grad=1e-10; narx_net.trainParam.Epochs=1000;
6. Configure and train the NARX ANN as result of the commands:
   [p,Pi,Ai,t]= preparets(narx_net,X,{},Y);
   narx_net=train(narx_net,p,t,Pi);
7. Close narx_net 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 2 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.

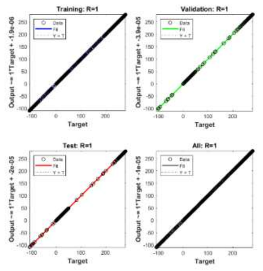
Since with NARX neural networks it is more frequent to get into overfitting situations, in this part of the study it was decided to manually divide the dataset into 70% for training, 15% to validate that the ANN is generalizing correctly and stop it before overfitting, and 15% to test the generalization 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 13.



(**a**)



(**b**)



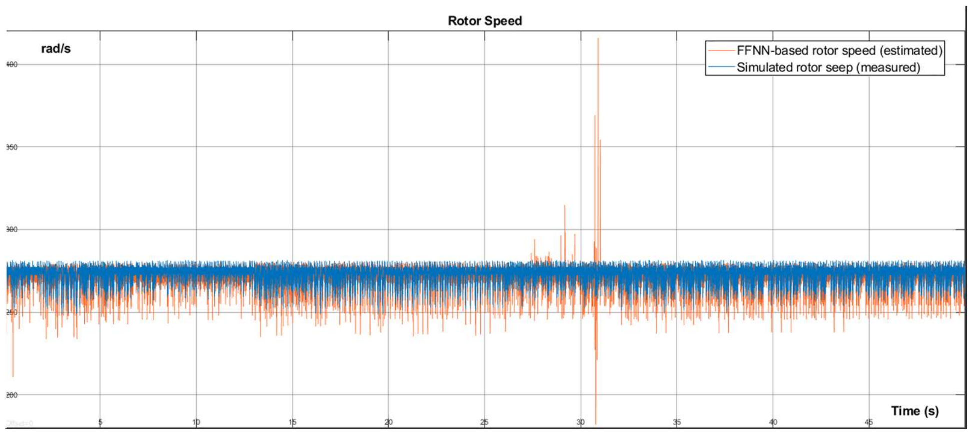
(**c**)

Figure 13. (**a**) Performance of the NARX-based rotor speed estimator considering mean squared error (MSE); (**b**) Error histogram of the NARX-based rotor speed estimator; (c) Regression between the rotor speed estimated by the NARX and the measured in the IM drive during training, validation and testing of the speed estimator.

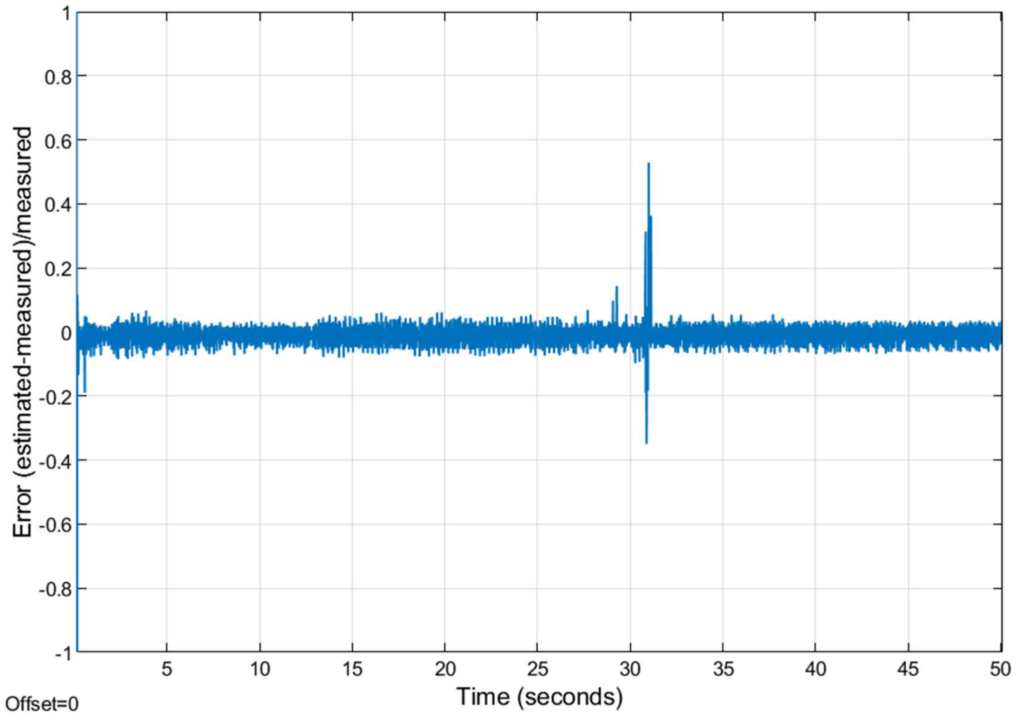
5.3 Results and Discussion

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. Figure 14 shows the measured speed with the IM driver model and the estimated speed based on the FF ANN. Figure 15 shows the actual reference speed of the IM driver and the speed estimated by the NARX ANN-based estimator. We can see 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 2 different types of ANN has been correctly validated.



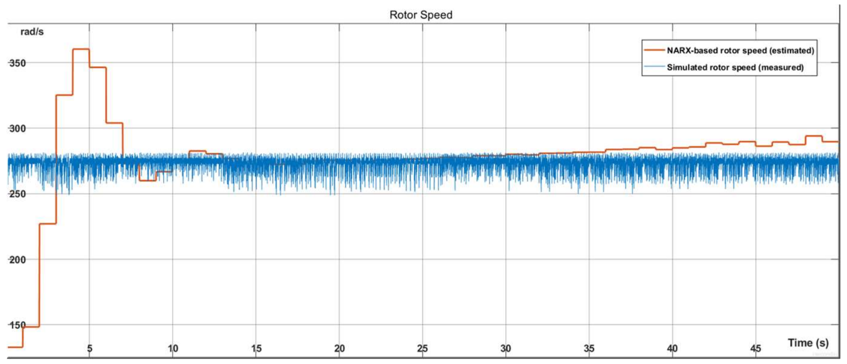
(**a**)



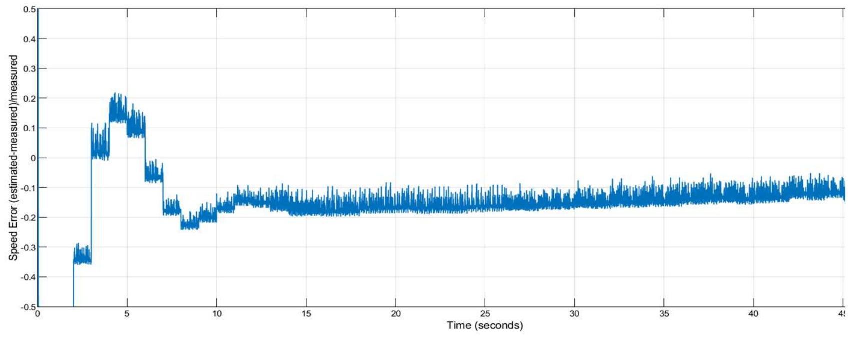
(**b**)

Figure 14. (**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).

However, in Figure 15 we can observe that perturbations in the response (ω_r) of 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 angular speed in the stator, since any change in the value 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 (b), 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, Figure 15(a) shows a perturbation of the rotor speed at 31 s that has no effect on the estimated output speed, which confirms the self-establishing capability of the first estimator.



(**a**)

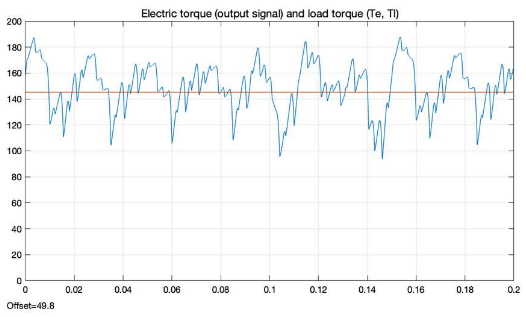


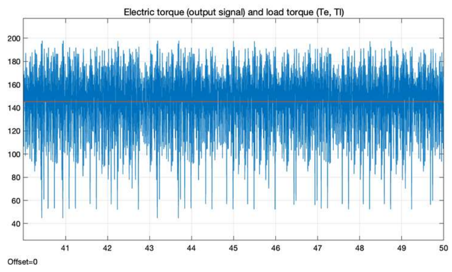
(**b**)

Figure 15. (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 ωr (scaled to 1.0).

6. ANN-based Electromagnetic Torque Estimator

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





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

Figure 16. (**a**) Zoomed electromagnetic torque (blue) M_e (Nm) measured with the Simulink model of the IM drive and load torque (orange)M_l (Nm) input to models during simulation; (b) Same measurements spanning the 50 s time run.

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 Me measured with the IM drive have been shown in Figure 16.

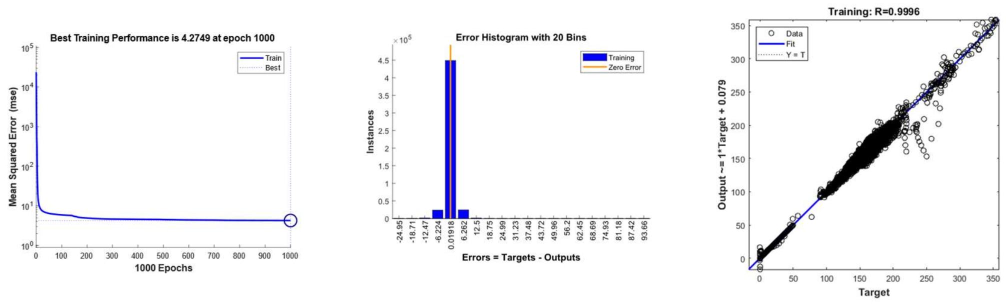
6.1 FF Network-based Electromagnetic Torque Estimator

The input data to the FF ANN estimator are based on the measured currents and voltages in the IM drive model of Figure 4, 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,

```
TqTT= timeseries2timetable(Torque);
TqT1= [seconds(TqTT.Time), TqTT.Data];
Tq= tonndata(TqT1,false,false);
W= [XT1(:,1:4),ZT1(:,2:4)];%(time, currents, voltages)
[ff_net, tr]= train(ff_net, W, Tq);
genism(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 17(a). The error histogram in Figure 17(b) shows that more than 99% of all the instances exhibit almost negligible error. (<0.020).



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

Figure 17. (**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 torque M_e estimated by the FF ANN and the measured in the IM drive.

It can be seen that there is a linear regression between the estimated and measured electromagnetic torque for the rotor with the IM driver. The regression has been calculated from the IM Simulink model and the results are presented in Figure 17(c).

6.2 NARX Network-based Electromagnetic Torque Estimator

A second NARX ANN-based estimator of electromagnetic torque M_e for the rotor has been generated with the DLT commands,

```
narx_net= narxnet(narx_net,delay1,delay2,10);
[x,xi,ai,t]= preparets(narx_net,W,{},Tq);
narx_net= train(narx_net, x, t, xi, ai);
narx_net_closed= closeloop(narx_net);
```

```
5.  %Simulink- NARX ANN block generated
6.  genism(narx_net);
```

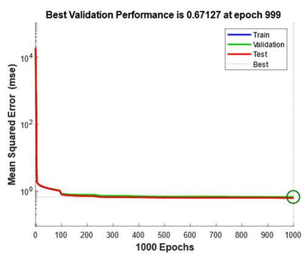
The time series $y(t)$ contains now values of Me and the series x(t) the vectors of the motor input currents and voltages in the 'abc' phase-normal reference system.

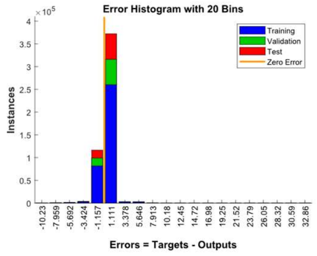
Table 3. Created NARX characteristics.

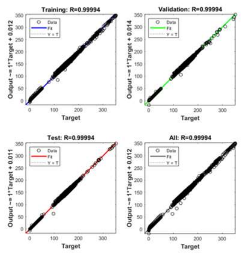
Series role	Time steps	Number of features	Features
Predictors: $x(t)$	250004	7	(time, i_a, i_b, i_c, v_a, v_b, v_c)
Responses: $y(t)$	250004	2 [1]	(time, M_e)

[1] Exogeneous signal components.

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 Figures 18 (a)-(c).







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

Figure 18. (**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 NARX-estimated and measured (Simulink IM model) electromagnetic torque Me during training, validation, and testing phases.

6.3 Obtained Results and Discussion

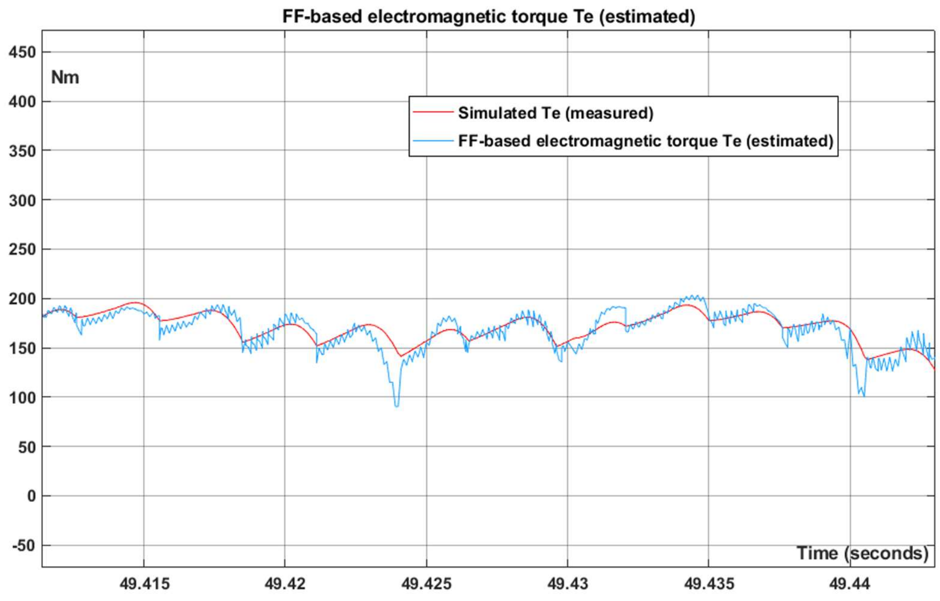
The DLT of MATLAB has been used to train, validate and testing the FF ANN and NARX electromagnetic torque estimator considering a time span of 50 s from the training speed and the load torque has been maintained constant and equal to 145 Nm in the plots.

Figure 19 (a) shows the zoomed capture of the measured data (used for training) and of 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 (Figure 4) with Simulink.

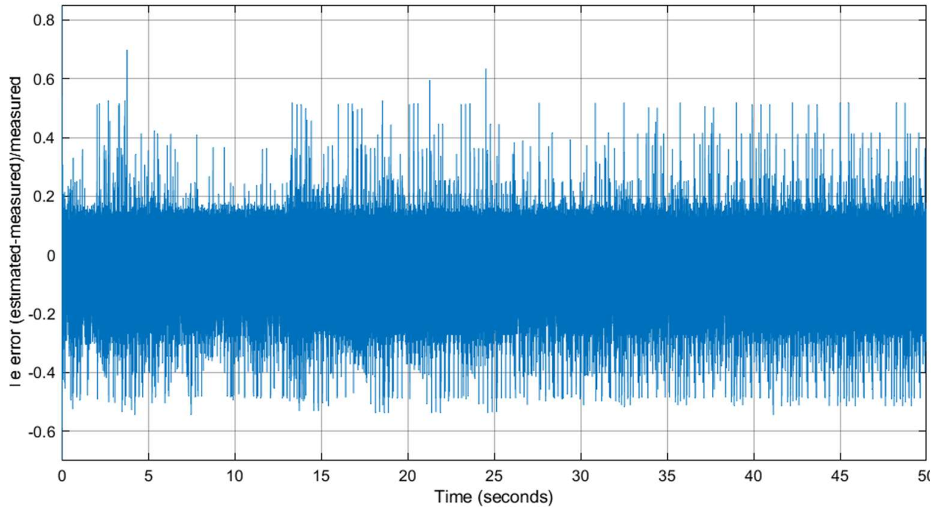
The errors, scaled to 1.0, obtained between the estimated and measured M_e for the rotor are shown in figures 19(b) and the results shown validate the accuracy and effectiveness of the FF ANN estimator implemented as a Simulink block by using DLT.

Figure 20 (a) 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 2 different types of ANN has been correctly validated.

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 20 (b). 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 attenuate and are 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.

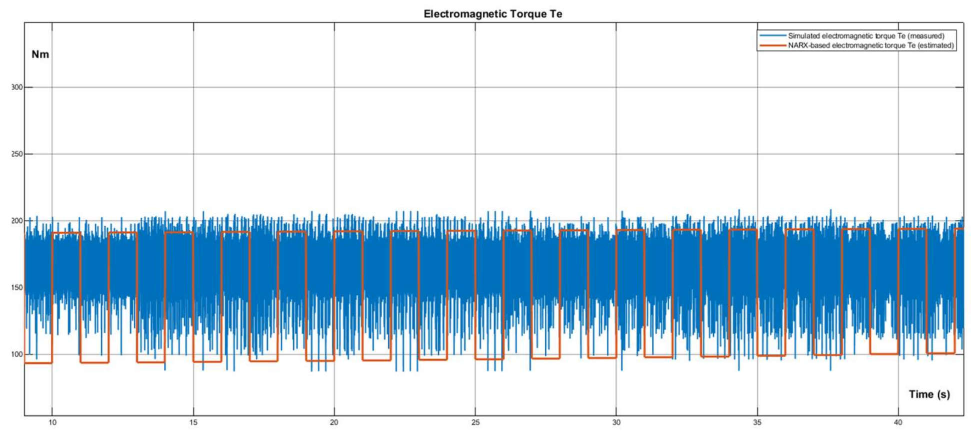


(**a**)

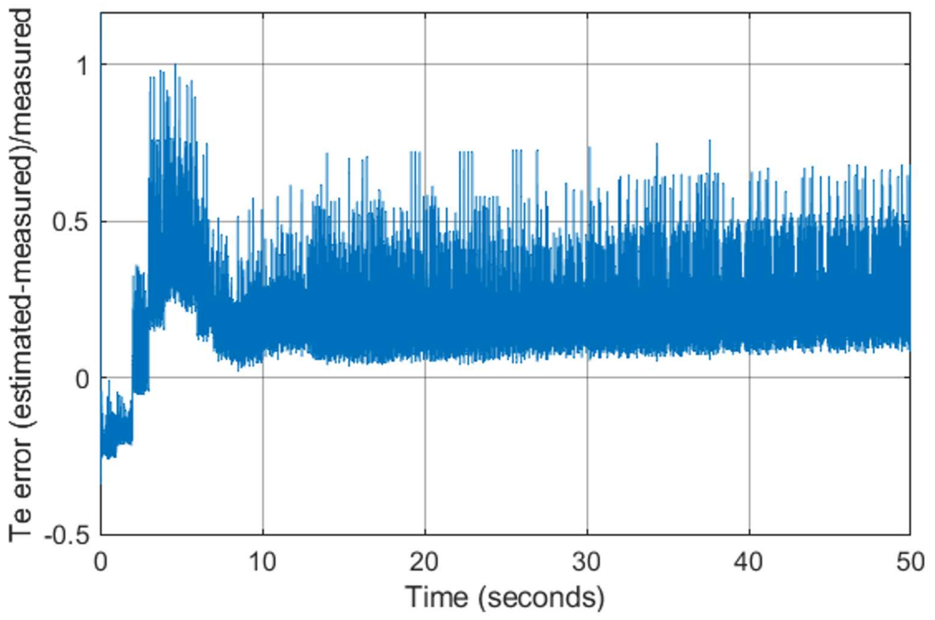


(**b**)

Figure 19. (**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 20. (**a**) Electromagnetic torque (orange) M_e -measured for training. M_e-estimated (blue) by the NARX ANN-based predictor; (**b**) M_e error (scaled to 1.0) between measured and estimated electromagnetic torque for the rotor.

7. 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 modelling continuous and discrete dynamic systems, such as the induction motor (IM) drive that is the case study.

Four estimators, based in a feed forward (FF) and a NARX ANN have been used for estimating 2 output signals: the rotor speed and electromagnetic torque 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) that provides a specific command for transforming certain types of ANNs in to 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 has been detailed as a useful reference for engineers and researchers. The estimated rotor speed and electromagnetic torque follow the same track than 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. 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 of the effectiveness and of the estimation accuracy with respect to the ANN deployed in this paper.

Finally, the proposed method has been defined for its easy integration in industrial environments for simulation (Simulink) and can also be used with standard libraries for neural networks development, such as SkLearn [11] 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. As future work, we plan to develop a tool capable of automated code generation of real-time and embedded system software for several computing platforms.

Funding: This study is funded by B-TIC-42-UGR20 through regional funds and when applicable cofounded by the Spanish Science Ministry grant PID2020-112495RB-C21.

Data Availability Statement: The complete IM Simulink model can be found at address https://lsi2.ugr.es/~mcapel/miscelanea/motor/.

Acknowledgments: I would like to thank prof. 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.

References

1. Mehrotra, P.; Quaicoe, J.E.; Venkatesan, R. Development of an artificial neural network based induction motor speed estimator. In *Proceedings of the PESC Record 27th Annual IEEE Power Electronics Specialists Conference*, Baveno, Italy, 23–27 June 1996, pp: 682-688.
2. Maclaurin, D.; Duvenaud, D.; Adams, R. Gradient-based Hyperparameter Optimization through Reversible Learning. In: Bach and Blei, 2015, pp. 2113–2122.
3. J.R.S. Iruela, L.G.B. Ruiz , M.I. Capel and M.C. Pegalajar. A TensorFlow Approach to Data Analysis for Time Series Forecasting in the Energy-Efficiency Realm. Journal Energies (MDPI). In *Energy Fundamentals and Conversion, Time Series Forecasting for Energy Consumption*-Special Issue, 2021, pp:1-22.
4. Maler, O.; Manna, Z.; Pnuelli,A. From timed to hybrid systems. In: *Proceedings of REX workshop* "Real-time: theory in practice", Springer-Verlag.1999, pp:1-37.
5. Franceschi, L., Donini, M., Frasconi, P., Pontil, M. (2017) Forward and Reverse Gradient-Based Hyperparameter Optimization. In*: Proceedings of the 34th International Conference on Machine Learning*, Sydney, Australia, PMLR 70, 2017, pp. 1165–1173.
6. Pedregosa, F.: Hyperparameter optimization with approximate gradient. In: *33rd International Conference on Machine Learning*, Balcan and Weinberger, New York, NY, USA, 2016, pp. 737–746
7. Almeida, L.; Langlois, T.; Amaral, J.; Plakhov, A. Parameter Adaptation in Stochastic Optimization. In D. Saad (Ed.), *On-Line Learning in Neural Networks* (Publications of the Newton Institute, pp. 111-134). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511569920.007, 1999, pp:111-134.
8. Baydin, A.G.; Cornish, R.; Rubio, D.M.; Schmidt; M., Wood, F. Online Learning Rate Adaption with Hypergradient Descent. In: *Proceedings of the International Conference on Learning Representations* (ICLR'18), 2018, pp:1-11.

9. Loshchilov, I.; Hutter, F. CMA-ES for hyperparameter optimization of deep neural networks. In: International Conference on Learning Representations Workshop track, 2016, pp: 1-5.
10. Cawley, G.; Talbot, N. On Overfitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research*, 2010, 11, pp:2079-2107.
11. Levesque, J.C. Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces. Ph.D. thesis, Université Laval, Quebec, Canada, 2018.
12. Feurer, M.; Klein, A.; Eggensperger; K., Springenberg; J.T., Blum; M., Hutter, F. Efficient and robust automated machine learning. In: *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems* (NeurIPS'15), Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.). 2015, pp. 2962–2970.
13. Pomerleau, F.; Colas, F.; Siegwart, R. et al. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 2013, 34, pp:133–148.
14. 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 Dataport*, 2020, https://dx.doi.org/10.21227/fmnm-bn95