ML Models FOR REAL-TIME HYBRID SYSTEMS

Manuel I. Capel
Department of Software Engineering, 18071University of Granada, Spain
manuelcapel@ugr.es

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 resorting to Machine Learning models, which can be understood as Automated Machine Learning for Real-time systems that present some degree of hybridation. An AC motor controller which must be able to maintain a constant air flow through a filter is one of these systems. The article also discusses a practical application of the method for implementing a closed loop control system to show how the proposed procedure can be applied to derive complete hybrid system designs with ANN.

Palabras clave: Automated Machine Learning, Real-time embedded control systems, Cyber-physical systems, Time series forecasting, Neural networks, Energy efficiency.

1 INTRODUCTION

Automated Machine Learning (AML) methods for design and implementation of Real-Time systems, applied to the specification of non-functional user requirements, such as timing constraints between system actions, are also starting to be applied successfully in the requirement specification, design and implementation phases of cyber-physical systems. Although AML methods help us to find a consistent implementation of system requirements, however 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 chose 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 expensive training phase of artificial neural networks (ANN) with hyperparameter updates carried out manually with the help of tools widely used in the industry (Simulink and Stateflow). The values of the hyperparameters can be validated following a mixed approach that is based on the interleaving of updates of the values with the training of the neural network. This method could be considered as very model-specific but, on the other hand, it allows us to obtain efficiently and with time constraints many hyperparameters of the cyber-physical model, which lays the basis for obtaining a great improvement over other approaches currently more used for example, obtaining hyperparameters using Bayesian optimization (HPO)[1].

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 [9] for Energy Efficiency systems; but if discrete components are interrelated with continuous ones, the common result is to produce inflexible models, with parameters that are difficult to change at run time in simulations.

In hybrid state machines [10] 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 now. In our case, we hypothesize that there is no major problem in building a trained ANN that substitutes the PID controller a closed loop control system to react and produce 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 remainder of the paper is structured as follows. We first give some background on Automated Machine Learning, which is the formal foundation of our method. Then, the approach proposed here is applied to solve an industrial problem of a real-time feedback closed loop used to maintain constant rotor speed of an induction motor driven by a *TriaC* device such as the one used by an AC motor to keep 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. Finally, the conclusions and the ongoing lines of work are presented.

https://doi.org/10.17979/spudc.9788497498043.752

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) implementation. There are no known mathematically well-founded algorithms for finding the best solution for RTS and for *control problems*, in general cyber-physical domains, within a limited amount of 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 do not guarantee that the exact optimal solution will be found, but they can lead to a near-optimal solution in a computationally efficient manner. Therefore, *metaheuristic methodologies* are gaining an every day 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,

- Can provide good solutions that can hinder traditional methods for computationally easy challenges involving large input complexity.
- Can yield sufficient solutions to difficult problems, e.g., problems for which an exact algorithm is not known and can be resolved in 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.
- Because metaheuristics can better deal with interties in goals, they are also able to recover from local optima.
- Most metaheuristics have only a few algorithmic changes to handle multiple objectives.

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 instructions' 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 artificial intelligence neural network (ANN**)** and thereof it must be considered as a computation model that is intrinsically parallel and convenient to solve very complex problems in the cyber-physical domain.

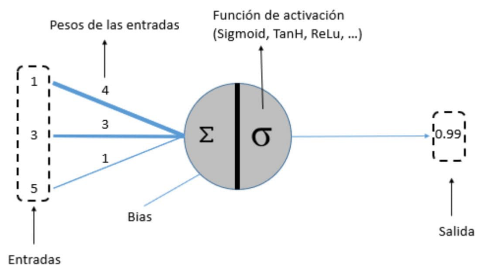

Figure 1: Artificial Neuron

Just as the human brain is made up of interconnected biological neurons, an ANN is made up of artificial neurons connected and grouped at different levels called layers. For a neural network 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.

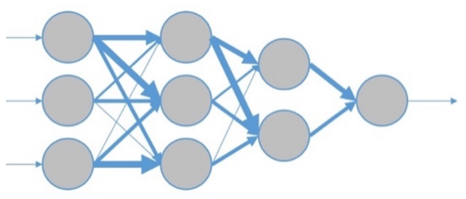

Figure 2: Simple Artificial Neural Network

2.1.1 Optimization and Gradient Descent

Modern parallel computers and graphics cards as NVIDIA GPUs make currently possible to obtain the gradient of model selection w.r.t. crucial hyperparameters of cyber-physical models [1].

https://doi.org/10.17979/spudc.9788497498043.752

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 [2] [3] by deploying gradient-based methods efficiently, more specifically, our method for ANN applied to high-dimensional HPO problems can perform the following tasks,

•optimize the learning rate of an ANN for each iteration and layer separately.

•Calculate optimal weights for each layer in the ANN.

•Reduce likelihood of overfitting by L2 regularization for each individual parameter of the model.

We can, therefore, overcome the backpropagation through the complete training procedure of an ANN by carrying out hyperparameter updates and by following separate validation 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 in order to enlarge the search space and then to haste select the right algorithm and its hyperparameters [4][5].

2.3 SCALABILITY

There are many machine learning problems which cannot be directly tackled due to their scale. We understand the term *scale* here as the size of the configuration space and the highly computational cost of individual model evaluations. There are currently some successes in training the neural network with small datasets and setting the hyperparameters of the training procedure manually [6]. 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 thousand 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 problem [7]. To reduce the amount of overfitting we can use a different shuffling for each function that we need to evaluate. 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 optima in the objective function, as it was noted in [8].

3 MODELING METHOD

In the proposed ANN-based method [9] 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 two main phases: *training* and *testing*. The generated output of the model, i.e., next values of variables of the cyber-physical system, which represent the functional and dynamic aspects of model, are used to feed Simulink/Stateflow blocks.

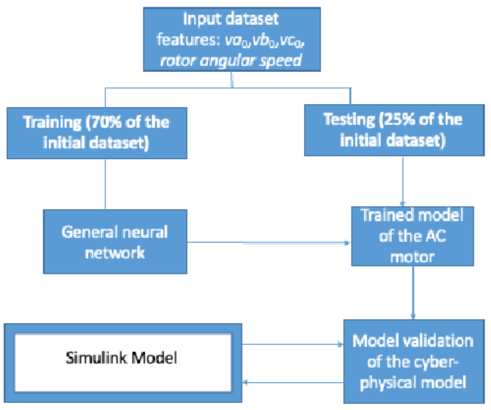
Input dataset
features: va_0, vb_0, vc_0, *rotor angular speed*
Training (70% of the initial dataset)
Testing (25% of the initial dataset)
General neural network
Trained model of the AC motor
Simulink Model
Model validation of the cyber-physical model

Figure 3: Diagram of the approach to deploy the AC Motor general model; there is only one neural network that learns from input data and outputs the next value

4. REGULATION OF ROTOR SPEED WITH AN INDUCTION MOTOR

An informal description of the user´s requirements specification of a closed loop control system is presented for controlling an AC motor (or induction motor) in figure 4. The open loop control of the engine is obtained by feeding it with a controlled voltage of 220 volts and 50 Hz. This control is carried out by cutting the sinus wave, which represents the input voltage using an electronic device named TriaC, which operates as a very fast switch. The control line of the TriaC is driven by a synchronization signal (*synch*), which informs when the input voltage passes through a zero value, at this moment the TriaC automatically stops to conduct electricity.

If after switching the TriaC off, it is fed with current several milliseconds later, it will be driven to saturation by the signal *texct* and will start to conduct until the input voltage passes through a zero value again. The closed loop of control is obtained in this case by calculating the precise time at which the TriaC must be enabled, so the excitation time must be calculated in real-time and in every cycle of the input voltage. The system should address its own safety if synchronization signal fails, or the TriaC overheats. If

https://doi.org/10.17979/spudc.9788497498043.752

the *synch* signal is missed out after passing a complete cycle of the input voltage, then *syncf* is raised. Other possible failure could happen if the TriaC overheated, in this case the electronic device might short-circuit and lead the engine to start working at the maximum number of revolutions, which would cause the loss of the engine after 1 second approximately.

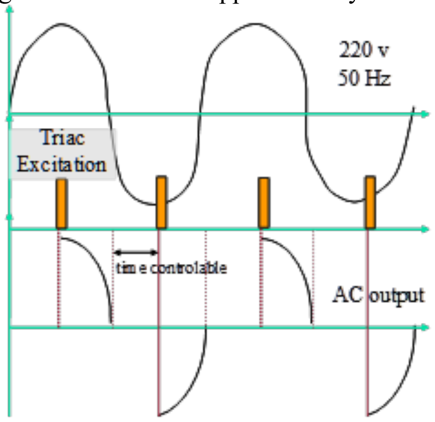

Figure 4: Description of the operation of *TriaC* device that controls the AC induction motor

The combination of induction motor with *TriaC* device can be used to control or maintain constant the velocity of a centrifuge of washing machine, the air flow through a filter, the speed of a vehicle, etc. From now on we suppose that AC induction motor directed by the *TriaC* device controls the air flow through a filter.

4.1 SYSTEM MODEL DERIVATION

The high-level diagram can be seen in the following figure 5, which includes five control flows: the synchronization signal (*synch*), which informs when the input voltage passes through zero value; the TriaC overheating warning; two signals, the first one signals the missing of *synch* and the second one signals TriaC overheating; the reactivating (*texct*) signals can make the TriaC returns to allow current to pass again. It also includes 2 data flows: the first one gives the present air flow through the filter (*flow*) and the second one, the air flow reference value (*ref*).

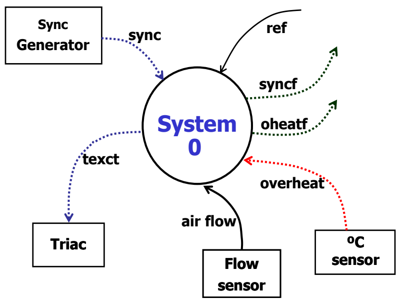

Figure 5. System Context Diagram.

value, respectively. If the *timeval* is outside the interval [0,1/freq*2], its value is saturated by the maximum or minimum value. Then one Simulink subsystem block can traditionally designed as the PID controller with its interface made up of two ports: the error signal (speed error signal) as the input port and the corrected *timeval* as the output one (figure 6).

4.2 MATLAB/SIMULINK MODEL OF AN INDUCTION MOTOR DRIVE

The model of an induction motor http://lsi.ugr.es/~mcapel/miscelanea/motor has been structured in 3 main blocks: (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); (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 give us the standard graphical representation of currents in the stator. Two specific blocks have been designed to calculate the electrical torque M_e given by the motor and another to calculate the rotor axis angular speed ω_r.

Finally, all the physical model constants given in table I have been defined using IS physical units in a *m-file* of Simulink, which has to be executed in Matlab before opening the Simulink model of the system.

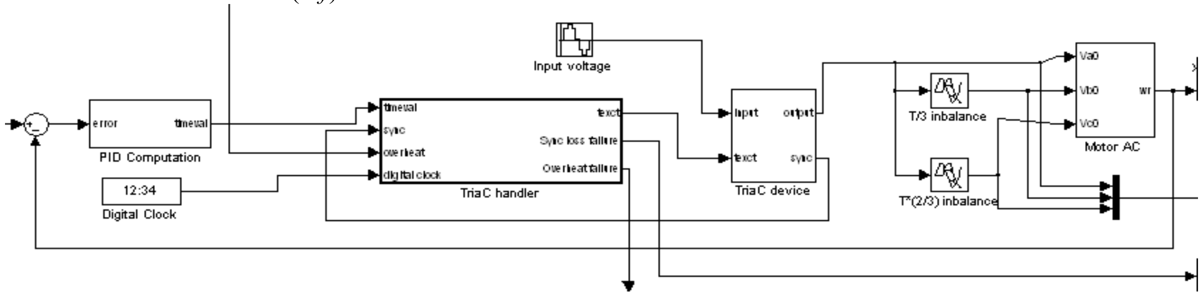

Fig.6. Simulink model of the final system

https://doi.org/10.17979/spudc.9788497498043.752

Table I: Constants and variables of the physical variables of an induction motor

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 referente 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: subíndex 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 electrical 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 $(\omega_e L_{ls})$	$\omega_b=2\cdot\pi\cdot f_b$: angular speed corresponding to the electric frequence of the motor feeding voltage.
χ_{lr}: rotor reactance $(\omega_e L_{lr})$	ω_r: rotor angular speed (output variable)

5. HYBRID SYSTEM SIMULATION

In order to carry out a simulation of the hybrid system proposed in this paper, more components must be added to the model. We need to construct the model of a *TriaC* device, one for the *AC Motor*, and the *sensor of flow* (i.e., a sensor to measure the speed of the rotor as a *tachometer* does in trucks), the *sensor of temperature* and one for the *sync generator*. This model is implemented by using the Simulink/ Stateflow framework, as figure 6 shows, as usual.

5.1 PHYSICAL MODELING OF AN INDUCTION MOTOR

The most difficult component to model is the *AC Motor* since the rest of the other devices can be modeled by means of simple switches or a combination of them.

The functioning of an induction motor is based on the Physical principle of mutual induction between electrical circuits traversed by a variable magnetic flux Φ. According to the Faraday law, which is given by the following equation:

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

the magnetic flux traversing a motor winding only depends on the current conducted by the circuit. It does not depend, for instance, on the number of poles of the motor. We can assign a self-induction constant *L* to any circuit being affected by magnetic induction, according to the equation:

$$N\cdot\Phi_B = L\cdot i_{reel} \qquad (2)$$

Nowadays the winding of induction motors is made of three windings, carrying each one of them a voltage phase separated π *rad*. from the next phase, which yields a rotating magnetic field in the stator, as figure 7 shows. The velocity of rotation is called the *synchronous speed*, which is given as a parameter of induction motors. If we short-circuit the rotor winding –using a squirrel cage winding, for instance-, then the motor will start rotating because the change in the magnetic field direction yielded by the synchronous speed of the stator ω_e induces a current that produces an electromagnetic force in the rotor. The difference between the rotation velocity of the stator ω_e and the rotor's one ω_r is named slip, which is also given as a parameter of induction motors.

5.2 THE TWO-PHASE SYNCHRONOUS ROTATING FRAME

We can assume a reference system that rotates at the synchronous speed ωe of the stator to ease the representation of the rotating $\vec{B}$ and inductance linkages by a system of coupled differential equations that describe the physical induction and motor dynamics, as figure shows. The induction motor is therefore modeled by two reels, the first one is aimed at conducting the current in the stator and it also generates the rotating magnetic field. The second one generates the induced magnetic field in the rotor. This simple model allows us to describe the magnetic coupling between the stator and rotor windings of an induction motor very accurately.

The first axis of the following figure is called the direct axis and the second one, the quadrature axis. θe is an additional variable representing the rotor angle and it can be considered an additional state of the induction motor model. An induction motor with a squirrel cage rotor winding (short-circuited) will have null v_{qr} and v_{dr} voltages. The constants and system variables of the above linear differential equations system are given in table I.

https://doi.org/10.17979/spudc.9788497498043.752

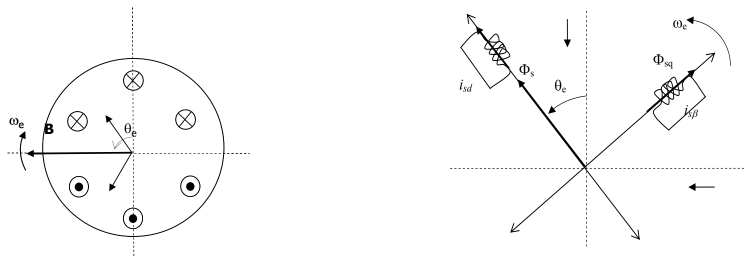

Figure 7: (a)Motor winding (b) Rotating reference system

5.3 CALCULATION OF THE ELECTROMAGNETIC TORQUE GENERATED BY THE MOTOR

We can derive a mathematical expression for calculating the electromagnetic torque generated by the motor. Since we know that the mechanical power given by the motor is obtained from the electromagnetic equation:

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

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

from the magnetic linkages F_{ds}, F_{qs}, and currents, which are obtained by solving a system of differential linear equations with concrete values of p (the number of poles) and ω_e as the input data to the induction motor model. The angular velocity ω_r of the rotor can also be calculated since the load torque T_l and moment of inertia J are also parameters of the induction motor model. As the above equations show, the electrical torque and the angular rotor velocity depend on the number of poles of the rotor winding, on the contrary of what happens with the magnetic couplings.

6. MATLAB/SYMULINK MODEL OF AN INDUCTION MOTOR DRIVE

The model of an induction motor http://lsi.ugr.es/~mcapel/miscelanea/motor has been structured in 3 main blocks: (1) transforms the three stator voltages v_a, v_b, v_c, with a phase of $2\pi/3$ 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); (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 give us the standard graphical representation of currents in the stator.

Two specific blocks have been designed to calculate the electrical torque M_e given by the motor and another to calculate the rotor axis angular speed ω_r. Finally, all the physical model constants given in *table I* have been defined using IS physical units in a m-file of Simulink, which must be executed in Matlab before opening the Simulink model of the system.

6.1 OBTAINED RESULTS

The results obtained with the two models (OLP and CLP) were quite different. In the first case, it was only considered an open control loop model; thus, only after a constant time the *TriaC* is excited in every cycle. In this case the disturbances in the system response (ω_r) are remarkable. Rotor speed follows the changes produced in the synchronous angular speed in the stator (ω_e) (figure 9), but any change in the value of ω_e provokes fast oscillations around the new value in the rotor velocity. If we carry out a simulation with the rotor velocity controlled by a PID, then we will obtain better results (figure 8).

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

Moreover, if we take a plot of the electrical torque output by the induction motor w.r.t. a constant load torque, which is given as an input variable to the system, we will take only important oscillations at the beginning, while the system is trying to get a stabilisation point. The oscillations shown in figure 8 represent about the 20% of the target value for the torque M_e, (300 Nm); these oscillations are caused by dynamic conditions during motor functioning, as the rotor axis friction.

https://doi.org/10.17979/spudc.9788497498043.752

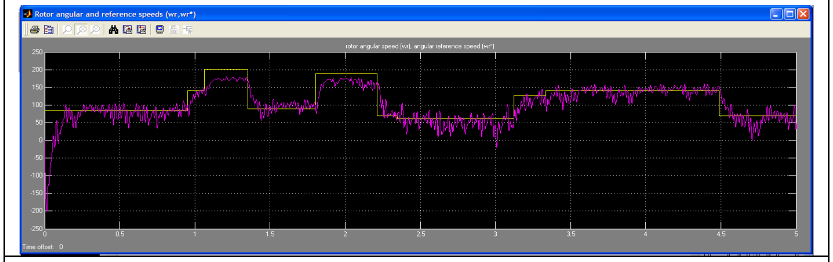

Figure 8: Closed loop control response to changes in the stator speed

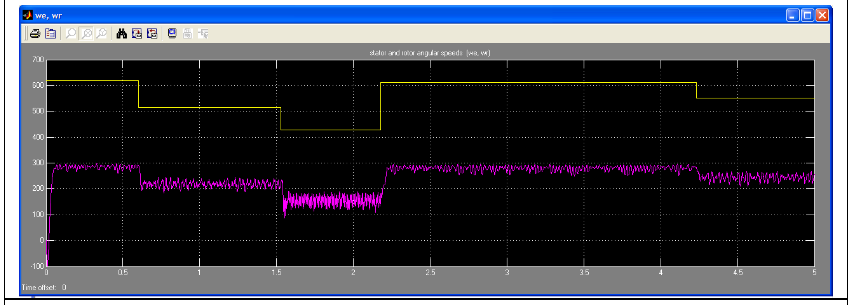

Figure 9: Response of the rotor to changes in the stator speed

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 modeling continuous and discrete dynamic systems, such as the AC motor controller that is the case study. We have shown that PID (*proportional integrative differential)* controllers can be substituted by a trained neural network that has been used 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 is mainly a set of guidelines at moment, which have proved to be of use for deriving a verifiable model of a cyber-physical complex system. The method has been defined for its easy integration in industrial environments for simulation (Simulink/Stateflow) and can be used with standard libraries for neural networks development, such as SkLearn [11] and PySpark. 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.

References

[1] Maclaurin, D., Duvenaud, D., Adams, R. (2015) Gradient-based Hyperparameter Optimization through Reversible Learning. In: Bach and Blei, pp. 2113–2122

[2] Franceschi, L., Donini, M., Frasconi, P., Pontil, M. (2017) Forward and Reverse Gradient-Based Hyperparameter Optimization. In: Precup and Teh, pp. 1165–1173

[3] Pedregosa, F.: Hyperparameter optimization with approximate gradient. In: Balcan and Weinberger, pp. 737–746

[4] Almeida, L.B., Langlois, T., Amaral, J.D., Plakhov, A. (1999) Parameter Adaptation in Stochastic Optimization, p. 111–134. Cambridge University Press

[5] Baydin, A.G., Cornish, R., Rubio, D.M., Schmidt, M., Wood, F. (2018) Online Learning Rate Adaption with Hypergradient Descent. In: Proceedings of the International Conference on Learning Representations (ICLR'18)

[6] Loshchilov, I., Hutter, F.(2016) CMA-ES for hyperparameter optimization of deep neural networks. In: International Conference on Learning Representations Workshop track

[7] Cawley, G., Talbot, N. (2010) On Overfitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. Journal of Machine Learning Research 11

[8] Levesque, J.C. (2018) Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces. Ph.D. thesis, Université Laval

[9] J.R.S. Iruela, L.G.B. Ruiz , M.I. Capel and M.C. Pegalajar (2021). A TensorFlow Approach to Data Analysis for Time Series Forecasting in the Energy-Efficiency Realm. Journal Energies

https://doi.org/10.17979/spudc.9788497498043.752

(MDPI). In Energy Fundamentals and Conversion, Time Series Forecasting for Energy Consumption-Special Issue (in press)

[10] Maler, O., Manna, Z., Pnuelli,A. (1992) From timed to hybrid systems. Proceedings of REX workshop ”Real-time: theory in practice”, Springer-Verlag.

[11] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F. (2015) Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS’15). pp. 2962–2970 (2015)

https://doi.org/10.17979/spudc.9788497498043.752