International Journal of Innovative Computing, Information and Control
Volume **15**, Number **2**, April **2019**

ICIC International ©2019 ISSN 1349-4198
pp. **445–464**

ADAMS-BASHFORTH NEURAL NETWORKS APPLIED IN A PREDICTIVE CONTROL STRUCTURE WITH ONLY ONE HORIZON

Paulo Marcelo Tasinaffo and Atair Rios Neto

Computer Science Division
Technological Institute of Aeronautics
Praça Marechal Eduardo Gomes, 50 Vila das Acácias, 12228-900 São José dos Campos/SP, Brazil
tasinaffo@ita.br; atairrn@uol.com.br

Received May 2018; revised September 2018

Abstract. *The neural networks application in control of dynamic systems always implies the need of training, to obtain an internal model of the system. One of the ways to represent the internal model of system dynamics is to design the neural network as a discretized model with delayed inputs through the NARMAX (Nonlinear Autoregressive Moving Average with eXogenous inputs) methodology. The network trained in this way has the disadvantage of requiring multiple neurons in the inner and input layers. Here, an alternative methodology for representing the dynamics of the system in a Multilayer Perceptron network is preliminarily tested. In this alternative methodology, using an Adams-Bashforth numerical integrator structure, allows a Multilayer Perceptron network architecture to be coupled to this integrator and to be designed to learn the functions of instantaneous derivatives in the ordinary differential equations of the dynamic system. The training structure can be called Adams-Bashforth neural network and is a particular case of a universal numerical integrator. This type of approach avoids unnecessary complexity in the network architecture and transforms its training in a problem of learning a static algebraic function. The application of this special neural network in a nonlinear predictive control structure is also developed. A practical example of orbit transfer control is considered to numerically test the proposed neural control methodology.*
Keywords: Nonlinear predictive control, Artificial neural networks, Dynamic systems modeling, Adams-Bashforth neural networks

1. **Introduction.** The nonlinear function mapping property is the central point for the use of neural networks in control. Training a neural network using input/output data from an invariant in time nonlinear plant can be considered as a problem of approximating a nonlinear function. It has already been shown that Multilayer Perceptron networks can arbitrarily approximate any continuous function (see [1-3]). A Multilayer Perceptron network with only a single inner layer is enough to represent any continuous function.

In the control literature, many well-established studies already exist (e.g., [4-7]) with emphasis on neural predictive control, for its efficiency and performance (e.g., [5-10]).

In this work the application of neural numerical integrators on a predictive control structure (see [11,12]) will be developed. Neural numerical integrators are a particular case of universal numerical integrators (e.g., [19,20]). The possibilities of using numerical integrators and artificial neural networks will be analyzed directly in strategies of nonlinear predictive control. This type of approach has the advantage of reducing the size of the neural network, and therefore facilitating its training, since the neural network only needs

DOI: 10.24507/ijicic.15.02.445

to learn the function of instantaneous derivatives of the original dynamic system (e.g, [11,12,19-21]).

It is important to note that the original work describing the Adams-Bashforth neural networks comes from [20,21]. However, in a complementary new way it is proposed here to apply the Adams-Bashforth neural networks in a predictive control structure. The predictive control structure proposed here will require the resolution of a nonlinear estimation problem to find the optimal control policy. This problem will be solved using Kalman filtering not only to train the feedforward network to represent the dynamics of physical systems (see [12]) but also to determine the smooth control policy in a mesh predictive control structure (see [10,22]).

To achieve the goals proposed here, this article is divided into the following sections. Section 2 briefly describes the methodology of the instantaneous derivatives. Section 3 mathematically develops the Adams-Bashforth neural network. Section 4 describes how to insert Adams-Bashforth neural networks into a predictive control structure, which will be solved with Kalman filtering. Section 5 presents a case study, the problem of two-dimensional orbit transfer between the planets Earth and Mars. Section 6 briefly describes the main conclusions of this paper.

2. **Instantaneous Derivatives Methodology.** In the methodology of the instantaneous derivatives (e.g., [19-21]), the universal approximator of functions will only play the role of a static function, that is, $\dot{y} = f(y,u) \cong \hat{f}\,(y,u,\hat{w})$, in the system of differential equations, and the final dynamics approximated through the use of some high-order numerical integration structure of single steps or multiple steps. Wang and Lin in [19] originally developed this methodology using a Runge-Kutta 4-5 single-step integrator coupled to a neural network with Radial Basis Functions (RBF) architecture. In this methodology, it is important to note that the training and simulation phases will basically depend on two approximation errors: one, due to the mean square error of the supervised training of the universal approximator; and the other, due to the order and integration step of the numerical integrator used. It is known that, in general, the greater the order of the integrator used and the smaller the integration step used, then the greater the precision obtained in each step of the simulation is. Figure 1 schematically illustrates the methodology of the instantaneous derivatives as proposed in [20,21] using a low order and high order Adams-Bashforth type multi-step integrator.

As can be seen in Figure 1, the great advantage of coupling a universal approximator of functions with a high-order numerical integrator structure is that the instantaneous derivative function, which governs the dynamics of the system considered, can be adapted to the input/output data from real-world problems. Without the use of a universal approximator, with only the use of a high-order numerical integrator it is impossible to adapt the dynamics of the system in question to real-world data. This impossibility can be visualized in Figure 1 through the variable $\vec{y}^{\,m'}(t+\Delta t)$.

The instantaneous derivative methodology has at least two advantages over the NARMAX methodology: the fact that the neural training is static and not dynamic (it is only necessary to learn the static function of the instantaneous derivatives) and that it allows the variation of the integration step in the simulation phase. The NARMAX methodology requires a new training if the integration step Δt is to be varied.

There are two forms of supervised training that the methodology of the universal numerical integrators allows to realize, depending on when it is known or when it is not known a priori the function of instantaneous derivatives of the dynamic system model. These two methodologies are (see Figure 2): the direct methodology and the empirical or indirect methodology.

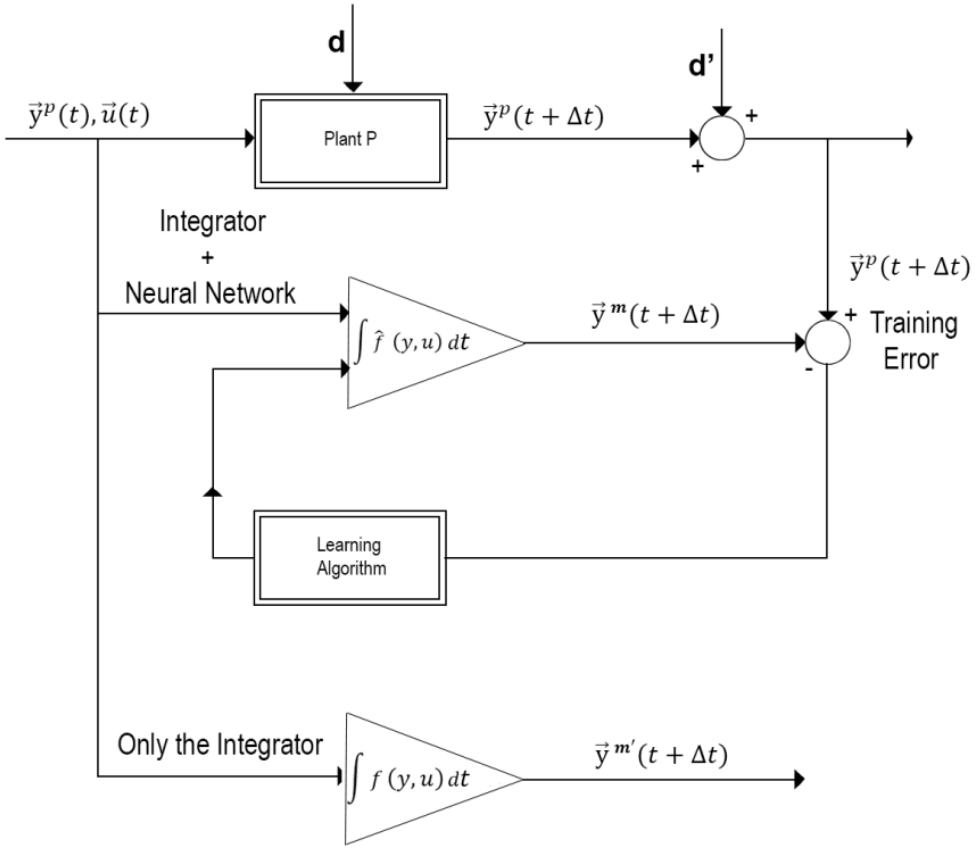

FIGURE 1. Illustrative diagram of the use of a universal approximator coupled to a high-order numerical integrator structure (source: see [12])

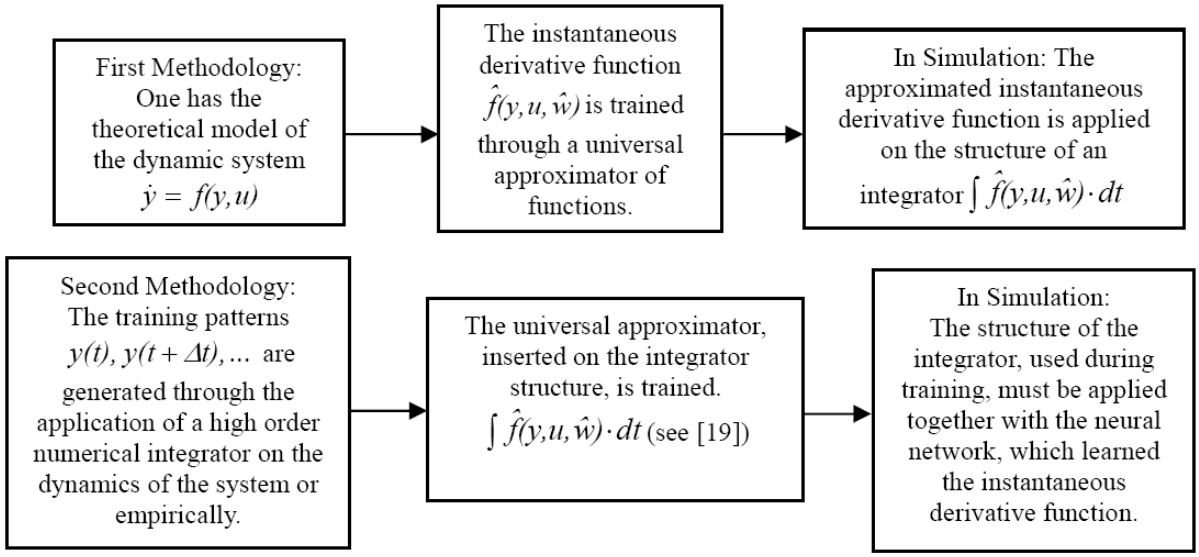

FIGURE 2. Two distinct ways of training a universal approximator in the structure of a numerical integrator

Direct Methodology. The first methodology is to simply randomly generate a sufficient set of values of the instantaneous derivative function within a domain of interest and directly train the universal approximator of functions to learn this derivative function (see [11]). After the training, the numerical simulation of the results can be done by applying

the approximated instantaneous derivative function on the structure of the integrator. For the evaluation of errors, the numerical solution of the integrator must be obtained by applying the two derivative functions, the trained and the original, so that the results of the solution propagation are compared. In this methodology, the numerical integrator will only be used in the simulation of the results. This methodology can only be applied if the mathematical model of the plant is known.

Empirical or Indirect Methodology. In the second methodology, the temporal solutions of the dynamic system $y(t), y(t + \Delta t), \ldots, y(t + n \cdot \Delta t)$ are generated from a sufficient number of initial conditions and control values through the structure of an integrator, preferably of high order to guarantee the high precision of the training patterns. The learning of the instantaneous derivatives function by the universal approximator of functions is only possible if the universal approximator is trained coupled to the structure of the numerical integrator (see [19-21]). The analysis of the error obtained by the integrator set and neural network is also performed by comparing the numerical solutions propagated by the original and trained instantaneous derivative functions. In this methodology, the numerical integrator is used in the generation of training patterns and is required during training and network simulation. This methodology can be used both for dynamic systems where the theoretical model is known as well as for real-world plants where the mathematical theoretical model of the system is not known but inputs and outputs can be empirically generated.

In this article, the methodology used to model the dynamic system will be the direct one. However, for a complete description of the indirect methodology (or empirical methodology) to train an Adams-Bashforth neural network see [20,21]. Unfortunately, these two references were originally written in Portuguese. For this reason, the next section will briefly detail how to train a fourth-order Adams-Bashforth neural network with empirical or indirect methodology, using the extended backpropagation algorithm.

3. **Adams-Bashforth Neural Networks.**

Artificial neural networks are considered universal approximators of functions as described in [1-3]. In this context, the technology of neural integrators that are neural networks coupled to numerical integration structures has been successfully developed. This section presents and develops an alternative empirical methodology to model and to obtain the instantaneous derivatives functions for nonlinear dynamic systems through a supervised training using numerical integrators of the multi-step Adams-Bashforth type, coupled to a universal approximator of functions. The particular case of the universal approximator being represented by a Multilayer Perceptron neural network will be considered. The structure to be developed here will be called the Adams-Bashforth neural network.

In this approach the neural network, coupled to the structure of the considered numerical integrator, plays the role of the instantaneous derivatives functions. The resulting neural integrator structure will be composed of a linear combination of feedforward neural networks with delayed responses. It is an important fact that it is only with the use of highest-order numerical integrators that the neural network can actually learn the instantaneous derivative functions with adequate accuracy, while in the case of first-order integrators it can only learn the mean derivatives (see [19-21,23-25]).

This approach is an alternative to the methodology that addresses the problem of neural modeling in high-order Runge-Kutta type simple-step integration structures, initially proposed by Wang and Lin in [19]. This latter approach has the drawback of being more complex when used in the backpropagation algorithm, which requires, in this case, the use of the chain rule for composite functions. This fact makes it very difficult to determine the partial derivatives required by the backpropagation algorithm.

Given the complexity of dealing with Runge-Kutta networks, other methodologies, in principle simpler than these, were developed. Among them are the application of simple and multiple step integrators developed in [12,20,21,23-25]. The instantaneous derivatives methodology is characterized by the use of variable integration step sizes, with integration structures of any order. However, in this methodology, precise results, allowing variation of the integration step, are only achieved with higher order integrators.

In this section it is proposed to empirically model the instantaneous derivatives functions by means of neural integrators that do not work with composite functions. This can be achieved with the Adams-Bashforth multi-step integrators, since they only use a linear combination of the instantaneous derivatives functions, facilitating the determination of the partial derivatives, required by the backpropagation algorithm, in the supervised training phase. Thus, this type of procedure avoids the excessive and costly use of the chain rule on the numerical integration structure.

Given the autonomous system of ordinary differential equations (ODE):

$$\dot{y} = f(y) \tag{1a}$$

where

$$y = [y_1, y_2, \ldots, y_{n_e}]^T \tag{1b}$$

$$f(y) = [f_1(y), f_2(y), \ldots, f_{n_e}(y)]^T \tag{1c}$$

the numerical resolution of the first order equation consists of calculating the value of the state variable in a discrete sequence of instants using the derivative function $f(y)$, which is the instantaneous derivative.

In the training of a feedforward network, learning by error correction is the most used technique to teach the network to approximate a specific task. In this learning, the vectors of the desired value ($\overline{y}$) and the output ($\hat{y}$), of a given network in the time t, are considered and the output error of the network can be represented by:

$$r^i(t) = \overline{y}(t) - \hat{y}^i(t) \tag{2a}$$

where

$$\overline{y}(t) = \left[\overline{y}_1(t), \overline{y}_2(t), \ldots, \overline{y}_{n_e}(t)\right]^T \tag{2b}$$

$$\hat{y}^i(t) = \left[\hat{y}^i_1(t), \hat{y}^i_2(t), \ldots, \hat{y}^i_{n_e}(t)\right]^T \tag{2c}$$

The variables i, t and n_e denote, respectively, the i-th iteration of supervised learning, the t-th training pattern ($t = 1, 2, \ldots, p$), and the total number of states. In learning by error correction, vector $r^i(t)$ triggers a control mechanism that produces a sequence of adjustments in the network parameters. The adjustments have the property of correcting, step by step, the output signal $\hat{y}^i$ with respect to the desired response $\overline{y}^i$ to $i = 1, 2, \ldots, I$ until the network achieves a desirable error.

This objective can be achieved, in general, by minimizing a cost function or performance index $J^i(t)$, given by the scalar product of Equation (3) expressed as follows:

$$J^i(t) = \frac{1}{2} r^i(t) \circ r^i(t) \tag{3}$$

For this quadratic functional the following relation is valid:

$$\frac{\partial J^l(t,i)}{\partial w^l_{jk}} = -r^i(t) \circ \frac{\partial \hat{y}^i(t)}{\partial w^l_{jk}} \tag{4}$$

In Equation (4), the index l represents the l-th network layer, the index k the k-th neuron of the previous layer $(l-1)$ and j the j-th neuron of the current layer l. It is also

possible to represent the partial derivatives of functional $J^i(t)$ as follows:

$$\frac{\partial J^l(t,i)}{\partial w_{jk}^l} = -\sum_{m=1}^{n_e} \left[\overline{y}_m(t) - \hat{y}_m^i(t)\right] \cdot \frac{\partial \hat{y}_m^i(t)}{\partial w_{jk}^l} \tag{5a}$$

or

$$\frac{\partial J^l(t,i)}{\partial w_{jk}^l} = -\left[\overline{y}(t) - \hat{y}^i(t)\right] \circ \frac{\partial \hat{y}^i(t)}{\partial w_{jk}^l} \tag{5b}$$

If $\hat{y}(t)$ is approximated by a multi-step neural integration structure, a linear combination of the conventional backpropagation can be applied to the same network or in different networks over the quadratic functional.

The analytical equation of the instantaneous derivatives for multiple-step integrators was deduced based on the backward chaining for ease of mathematical elaboration, according to Equation (6), and it is important to note that the order o of the integrator is also the total number of linear combinations to feedforward networks to be employed. By definition, $t_n = t + n \cdot \Delta t$. In this way, one has:

$$\hat{y}^i(t_n) = y^i(t_{n-1}) + \frac{h}{\alpha} \cdot \sum_{i=1-o}^{0} \beta_i \cdot \hat{f}[y(t_{n-1-i})] \tag{6}$$

where α and β_i's are the integration coefficients as a function of order o of the numerical integrator; h is the integration step; $\hat{f}[\ldots]$ is the output of the neural network; $\hat{y}(\cdot)$ is the model of the dynamic system. Substituting Equation (6) into (5b):

$$\frac{\partial J^l(t_n,i)}{\partial w_{jk}^l} = -\left[\overline{y}(t_n) - \hat{y}^i(t_n)\right] \circ \frac{\partial \left\{ y^i(t_{n-1}) + \frac{h}{\alpha} \cdot \sum_{i=1-o}^{0} \beta_i \cdot \hat{f}[y(t_{n-1-i})] \right\}}{\partial w_{jk}^l} \tag{7a}$$

or

$$\frac{\partial J^l(t_n,i)}{\partial w_{jk}^l} = -\left[\overline{y}(t_n) - \hat{y}^i(t_n)\right] \circ \left\{ \frac{h}{\alpha} \cdot \sum_{i=1-o}^{0} \beta_i \cdot \frac{\partial}{\partial w_{jk}^l} \hat{f}[y(t_{n-1-i})] \right\} \tag{7b}$$

It is known that the scalar product has the following properties:

1. Commutative law: $A \circ B = B \circ A$
2. Distributive law: $A \circ (B + C) = A \circ B + A \circ C$
3. For m a scalar variable: $m(A \circ B) = (mA) \circ B = A \circ (mB) = (A \circ B)m$

Combining properties (2) and (3):

$$A \circ (\alpha B + \beta C) = \alpha(A \circ B) + \beta(A \circ C) \tag{8}$$

At the level of comparison, for the aspect of algorithmic complexity, the following expression (9a) defines the conventional backpropagation, while Equation (9b) is the application of property (8) in (7b), thus resulting in extended backpropagation. It will be seen that the use of Adams-Bashforth neural integration structure of order o will result in the linear combination of o delayed backpropagations during the neural training phase.

$$\frac{\partial J^l(t_n,i)}{\partial w_{jk}^l} = -\left[\overline{y}(t_n) - \hat{y}^i(t_n)\right] \circ \frac{\partial}{\partial w_{jk}^l} \hat{f}[y(t_n)] \tag{9a}$$

$$\frac{\partial J^l(t_n,i)}{\partial w_{jk}^l} = -\frac{h}{\alpha} \cdot \sum_{i=1-o}^{0} \beta_i \cdot \left[\overline{y}(t_n) - \hat{y}^i(t_n)\right] \circ \frac{\partial}{\partial w_{jk}^l} \hat{f}[y(t_{n-1-i})] \tag{9b}$$

Equation (9b) can be used to determine the Jacobian matrix $\frac{\partial J^l(t_n,i)}{\partial w_{jk}^l}$, where n_l is the total number of neurons in layer l, and n_{l-1} is the total number of neurons in the layer

$l-1$, as described in Equation (10).

$$\left[\frac{\partial J^l(t,i)}{\partial w_{jk}^l}\right]_{n_l\times n_{l-1}} = \begin{bmatrix} \dfrac{\partial J^l(t,i)}{\partial w_{11}^l} & \cdots & \dfrac{\partial J^l(t,i)}{\partial w_{1n_{l-1}}^l} \\ \vdots & \cdots & \vdots \\ \dfrac{\partial J^l(t,i)}{\partial w_{n_l1}^l} & \cdots & \dfrac{\partial J^l(t,i)}{\partial w_{n_ln_{l-1}}^l} \end{bmatrix} \tag{10}$$

The backpropagation or gradient algorithm only requires the feedforward network weights to be updated recursively through expression (11), where $l = 1, 2, \ldots, s$, for i equal to the i-th iteration of the gradient algorithm and s the total number of layers in the network.

$$\overline{W}_{i+1}^l = \overline{W}_i^l - \alpha \cdot \frac{\partial J^l(t,i)}{\partial \overline{w}^l} \tag{11}$$

Thus, if the linear combination of four delayed inputs on the neural network F_{n-4}, F_{n-3}, F_{n-2} and F_{n-1} is used in the calculation of the fourth-order integrator, there come results (see [15]):

$$y(t_n) = y(t_{n-1}) + \frac{h}{24} \cdot [55 \cdot F_{n-1} - 59 \cdot F_{n-2} + 37 \cdot F_{n-3} - 9 \cdot F_{n-4}] \tag{12}$$

Figure 3 presents a graphical scheme to represent fourth-order Adams-Bashforth neural networks. This figure shows the graphical scheme for making the linear combination of the integrator structure with four delayed inputs, which in this case are the four simultaneous inputs on the same feedforward network.

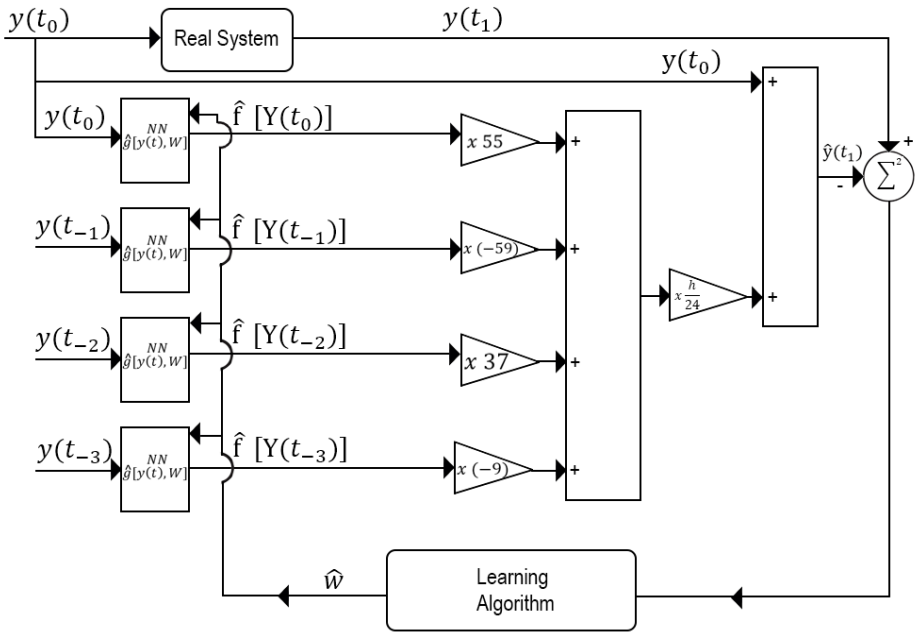

FIGURE 3. Graphic scheme of a fourth-order Adams-Bashforth neural integrator (source: see [20,21])

4. **Classification of Main Neural Control Structures.** Basically, there are three control structures, which can be used in conjunction with a universal numeric integrator. These structures are: IMC (Internal Model Control) structure, predictive control structure and adaptive control structure. In this section, only the predictive control structure is presented, which is the methodology effectively used in this article.

4.1. **Predictive control.** In what follows, adopting a heuristic and theoretical approach, the design and convergence analysis of a neural predictive control methodology will be presented. This methodology consists in solving an optimization problem with a quadratic index of performance, linking the already trained network with the desired dynamic system (see [5-10]). Because it is a nonlinear optimization problem, the solution is obtained iteratively for the discrete control actions through successive linearization. For the method to become feasible in practice the estimation of the controls must be obtained in real time. With the evolution of the current processors this is not a problem to prevent its use.

It can be demonstrated (see [10,12]) that Kalman filtering algorithms provide solutions that converge to soft solutions to control variables and at the same time track the reference trajectory. Unlike an IMC control structure, in a predictive control scheme it is not necessary to employ the inverse dynamics of the plant, and thus a neural training is avoided. On the other hand, in neural predictive control it is necessary to solve an optimization problem involving the neural model of the plant.

Figure 4 (see [10,11]) presents a simplified schema of the predictive control structure. As can be seen, a neural network is placed in parallel with the plant, in order to learn it. When the achieved learning is within an acceptable error or tolerance, the determination of the smooth controls that will trace the reference trajectory $r(t)$ can be obtained by the Kalman filtering algorithm, as the solution of a network-linked optimization problem.

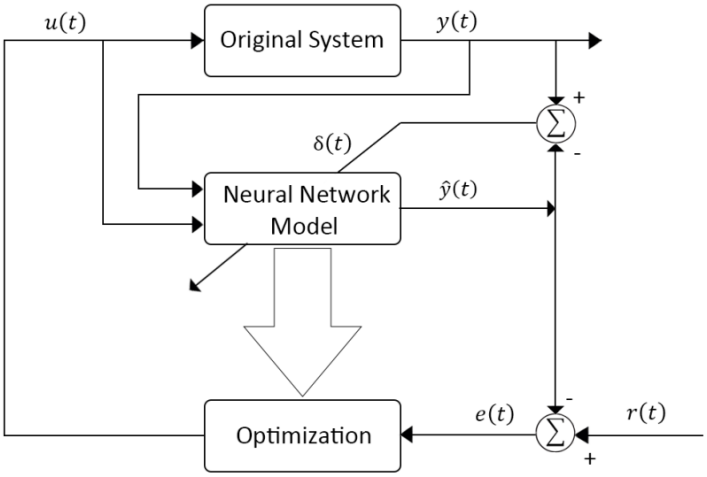

FIGURE 4. Neural optimization scheme for the determination of control which will track the reference trajectory (source: see [10,11])

In the scheme of Figure 4, the problems associated with feedforward neural network training and the determination of the soft control policy are both seen and treated in an integrated manner as stochastic nonlinear parameter estimation problem. The type of approach elaborated here allows to see the optimal control problem in a more general stochastic structure and to derive versions of control algorithms that are formally equivalent to the versions of the Kalman filtering derived and used for the training problem of the feedforward neural network.

The problem to be solved is to control the dynamic system given by,

$$\dot{y} = f(y, u) \tag{13}$$

where a time-discrete nonlinear input/output model is used to predict approximate responses, given by:

$$\hat{y}(t_j) = f\left[y\left(t_{j-1}\right), \ldots, y\left(t_{j-n_y}\right); u\left(t_{j-1}\right), \ldots, u\left(t_{j-n_u}\right), w\right] \tag{14a}$$

for

$$t_j = t + j \cdot \Delta t \tag{14b}$$

The predictive neural control scheme uses a feedforward network with the capability to accurately learn a mapping like that represented by (14a) to model the dynamic system of Equation (13). The internal model represented by the network will then be the response model that can be used to determine the soft control actions that will trace the reference trajectory by minimizing a predictive quadratic performance index. The performance or functional index of this control structure is given by (e.g., [10,12]):

$$\begin{aligned} J(t) = \Bigg[& \sum_{j=1}^{n} \left[y_r(t_j) - \hat{y}(t_j) \right]^T \cdot r_y^{-1}(t) \cdot \left[y_r(t_j) - \hat{y}(t_j) \right] \\ & + \sum_{j=0}^{n-1} \left[u(t_j) - u(t_{j-1}) \right]^T \cdot r_u^{-1}(t) \cdot \left[u(t_j) - u\left(t_{j-1}\right) \right] \Bigg] \Big/ 2 \end{aligned} \tag{15}$$

where $y_r(t_j)$ is the reference trajectory at time t_j; n is the horizon on which control actions and reference trajectories are considered; $r_y^{-1}(t_j)$ and $r_u^{-1}(t_j)$ are the matrices of positive definite weights and $\hat{y}(t_j)$ is the output of the feedforward network trained to approximate the model of the dynamic system.

For more details on the construction of the functional of Equation (15) see [5,6]. The output of the feedforward network $\hat{y}(t_j)$ is represented by the expression (14a). The parameters or weights w of this network must have already undergone a training that produces an output in the network with an error within an acceptable tolerance. The first term of the functional of Equation (15) is associated with tracing the reference trajectory and the second term in determining a smooth control policy. When this functional is minimized these two conditions are expected to be satisfied simultaneously. Note that in the predictive control scheme, in a typical cycle, when this functional $J(t)$ is optimized, only the control determined at instant t is implemented; and so on, repeating the cycle for each subsequent discrete instant of time. The problem of determining the predictive control actions can also be treated as an optimal nonlinear estimation of parameters thus allowing the derivation and use of a Kalman filtering algorithm.

This stochastic method initially assumes that the problem of determining the control in the functional of Equation (15) can be seen as the following stochastic parameter estimation problem (see [10]):

$$y_r(t_j) = \hat{y}(t_j) + v_y(t_j) \tag{16a}$$

$$0 = u(t_{j-1}) - u(t_{j-2}) + v_u(t_{j-1}) \tag{16b}$$

$$E[v_y(t_j)] = 0,\ E\left[v_y(t_j) \cdot v_y^T(t_j)\right] = r_y(t_j) \tag{16c}$$

$$E[v_u(t_j)] = 0,\ E\left[v_u(t_j) \cdot v_u^T(t_j)\right] = r_u(t_j) \text{ for } j = 1, 2, \ldots, n \tag{16d}$$

where $\hat{y}(t_j) = f\left[y(t_{j-1}), \ldots, y(t_{j-n_y}); u(t_{j-1}), \ldots, u(t_{j-n_u}), w\right]$ is the output of the neural network; $\hat{y}(t_{j-1}), \ldots, \hat{y}(t_{j-n_y})$ and $u(t_{j-1}), \ldots, u(t_{j-n_u})$ are, respectively, the responses of the system and the control actions already taken and known; $v_y(t_j)$ and $v_u(t_j)$ are the uncorrelated components of noise for different values of t_j.

Equation (16a) states that the reference trajectory of the system state $y_r(t_j)$ at the future instant is equal to the estimation done by the network in relation to the delayed instants plus an uncertainty $v_y(t_j)$. Equation (16b) translates the smooth characteristic of control actions, i.e., two successive actions must be estimated in such a way that the difference between them is as close as possible to the zero mean of error $v_u(t_{j-1})$.

Equation (16b) is a recursive relation that can be expressed in the form of a priori information as follows:

$$\hat{u}(t_{-1}) = u(t_{j-1}) + \sum_{k=0}^{j-1} v_u(t_k) \tag{17}$$

where the a priori value $\hat{u}(t_{-1})$ is the estimated value of a control at time $t_{-1} = t - \Delta t$. In this way, the minimization of the functional given by Equation (15) is modeled as satisfying the observations of Equation (16a) subject to a priori information of Equation (17). The first consequence of treating this problem in a more general stochastic structure is that the weight matrices present in the objective function of Equation (15), i.e., $r_y^{-1}(t_j)$ and $r_u^{-1}(t_j)$, now have the meaning of covariance matrices associated with random variables whose standard deviations respectively model the precision in the tracking of the reference trajectory and the dispersion of the increment of the smooth control. These observations make the understanding of these definitions much easier.

To treat the problem represented by Equations (16a) and (16b) as a linear parameter estimation it is necessary to take a linearized approximation of Equation (16a) as follows:

$$\alpha(i) \cdot [y_r(t_j) - \overline{y}(t_j, i)] = \sum_{k=0}^{j-1} [\partial \hat{y}(t_j)/\partial u(t_k)]_{\{\overline{u}(t_k;i)\}} \cdot [u(t_k, i) - \overline{u}(t_k, i)] + v_y(t_j) \tag{18}$$

where k starts from zero, since $\hat{y}(t_j)$ is also a function of $u(t_{j-2}), \ldots, u(t)$ through successive recursions, starting with $\hat{y}(t_{j-1}), \ldots, \hat{y}(t_{j-n_y})$; $0 < \alpha(i) \leq 1$ is a constant that must be adjusted to guarantee the hypothesis of approximation of the linear perturbation.

The partial derivatives that appear in Equation (18) are calculated using the rule of backpropagation in relation to the outputs of the neurons of the feedforward network. In addition:

$$\begin{aligned} &\alpha(i) \cdot [\hat{u}(t_{-1}) - \overline{u}(t_l, i)] = [u(t_l, i) - \overline{u}(t_l, i)] + \sum_{k=0}^{l} v_u(t_k) \\ &\text{for } l = 0, 1, \ldots, n-1 \text{ and } i = 1, 2, \ldots, I \end{aligned} \tag{19}$$

where $\hat{u}(t_{-1})$ is the estimated solution of the last control step; $\alpha(i) \leftarrow \alpha(i+1)$; $\overline{u}(t, i+1) = \hat{u}(t_l, i)$ is the estimated approximate value of $u(t_l)$ in the ith iteration and for $i = 1$ the estimated or extrapolated values of the last control step are used.

For $j = 1, 2, \ldots, n$ and $l = 0, 1, \ldots, n-1$ the problem of Equations (18) and (19) is a linear stochastic estimation of parameters. This problem can be represented in a more compact and easier to understand notation:

$$U_l(t, i) \equiv u(t_l, i) \tag{20a}$$

$$\hat{U}_l(t_{-1}) \equiv \hat{u}(t_{-1}) \tag{20b}$$

Thus, the problem can be equivalently expressed as:

$$\alpha(i) \cdot \left[\hat{U}(t_{-1}) - \overline{U}(t, i)\right] = U(t, i) - \overline{U}(t, i) + V_u(t) \tag{21a}$$

$$\alpha(i) \cdot \overline{Z}^u(t, i) = H^u(t, i) \cdot \left[U(t, i) - \overline{U}(t, i)\right] + V_y(t) \tag{21b}$$

The meaning of the compact variables in the immediately preceding equations is obtained by direct comparison of Equations (21a) and (21b) with Equations (18) and (19), respectively. The Kalman filtering expressions in an ith iteration result:

$$\begin{aligned} \hat{U}(t, i) = \overline{U}(t, i) &+ \alpha(i) \cdot \left[\hat{U}(t_{-1}) - \overline{U}(t, i)\right] \\ &+ k(t, i) \cdot \alpha(i) \cdot \left[\overline{Z}^u(t, i) - H^u(t, i) \cdot \left[\hat{U}(t_{-1}) - \overline{U}(t, i)\right]\right] \end{aligned} \tag{22a}$$

$$k(t,i) = R_u(t) \cdot H^{u^T}(t,i) \cdot \left[H^u(t,i) \cdot R_u(t) \cdot H^{u^T}(t,i) + R_y(t)\right]^{-1}$$

$$\equiv \left[R_u^{-1}(t) + H^{u^T}(t,i) \cdot R_y^{-1}(t) \cdot H^u(t,i)\right]^{-1} \cdot H^{u^T}(t,i) \cdot R_y^{-1}(t) \quad (22b)$$

$$\overline{U}(t,i+1) = \hat{U}(t,i);\ \alpha(i) \leftarrow \alpha(i+1);\ \hat{U}(t) = \hat{U}(t,I) \quad (22c)$$

$$\hat{R}_u(t,I) = [I_u - K(t,I) \cdot H^u(t,I)] \cdot R_u(t) \text{ for } i = 1,2,\ldots,I \quad (22d)$$

where $R_u(t)$, $R_y(t)$ and $\hat{R}(t,I)$ are, respectively, the covariance matrices of $V_u(t)$, $V_y(t)$ and $\left(\hat{U}(t,I) - U(t)\right)$, and I_u is the identity matrix. The control calculated with this algorithm is the minimum of the functional:

$$\begin{aligned} J(\alpha,i) = & \left[\alpha(i) \cdot \overline{Z}^u(t,i) - H^u(t,i) \cdot \left[U(t,i) - \overline{U}(t,i)\right]\right]^T \cdot R_y^{-1}(t) \\ & \cdot \left[\alpha(i) \cdot \overline{Z}^u(t,i) - H^u(t,i) \cdot \left[U(t,i) - \overline{U}(t,i)\right]\right] \\ & + \left[U(t,i) - \overline{U}(t,i) - \alpha(i) \cdot \left[\hat{U}(t_{-1}) - \overline{U}(t,i)\right]\right]^T \cdot R_u^{-1}(t) \\ & \cdot \left[U(t,i) - \overline{U}(t,i) - \alpha(i) \cdot \left[\hat{U}(t_{-1}) - \overline{U}(t,i)\right]\right] \end{aligned} \quad (23)$$

Thus, the convergence to a smooth control $\hat{U}(t)$ that will track the reference trajectory $y_r(t)$ is guaranteed since the feedforward neural network $\hat{y}(t)$ has the capacity to represent the system dynamics of Equation (13) and to allow a linearized approximation in an ith typical iteration, when a sufficiently small value is used for $\alpha(i)$.

Since it is possible to represent dynamic systems using neural networks in a numerical integrator structure, it remains to know how to design this scheme on a predictive control structure. In the present case, the chain rule on the integrator structure combined with the backpropagation of the neural network should be used. Thus, the greater the order of the integrator is, the more complex the calculation of these derivatives will become.

In [12,22] a mathematical model is developed that combines the method of the mean derivatives, in a predictive control structure, for the general case, considering n horizons in the functional of Equation (15).

4.2. **The computational algorithm combining the Adams-Bashforth neural networks and the predictive control structure.** In a predictive control structure, the parameters to be estimated are the controls themselves that will keep the dynamical system around a reference trajectory within a desired control horizon. In this section a Kalman filtering algorithm is presented, with the aim of facilitating the computational implementation of this problem, which is one of a dynamic nonlinear programming type. Since it is difficult to construct a generic algorithm, particular values will be assumed for n, n_y and n_u. In this case, it will be adopted $n = 1$ (estimation horizon of control variable) and $n_y = n_u = 4$ (number of delayed inputs of the state and control variables of the neural integration structure).

As the proposed estimation problem is non-linear, then the algorithm will have an iterative characteristic, that is, it starts from an a priori information of the control variables, and then new values of these variables are recursively estimated until the output of the states of the system obtained by the neural network converges to the known reference trajectory, within an acceptable error for that horizon.

Note that the values of the state variables $y(t_{-3})$, $y(t_{-2})$, $y(t_{-1})$ and $y(t_0)$, and the values of the control variables $u(t_{-3})$, $u(t_{-2})$, $u(t_{-1})$ and $u(t_0)$ must be known as well as the reference trajectory and a priori information of the actuation controls within the

desired forecasting horizon. Thus, the algorithm for solving the proposed problem can be summarized in the following essential steps.

1) Generate the a priori information of the controls $u(t)$ within the desired horizon. As the desired control policy should be smooth as imposed by the functional J of Equation (15) nothing more natural than starting it with the null vector, that is,

$$\overline{U}(t,i) = [\overline{u}(t_0,i)]^T \equiv 0 \text{ for } I = 1 \tag{24}$$

where $\overline{u}(t_0,i)$ is the vector of the control variables at time t_0 in the i-th iteration.

2) Calculate the partial derivatives $\frac{\partial \hat{y}(t_j)}{\partial u(t_k)}$ present in Equation (18) through the expressions from (37a) to (37f) of the backpropagation algorithm (see Appendix A). Table 1 illustrates the particular case for $n = 1$ and $n_y = n_u = 4$.

$$\alpha(i) \cdot [y_r(t_j) - \overline{y}(t_j,i)] = \sum_{k=0}^{j-1} [\partial \hat{y}(t_j)/\partial u(t_k)]_{\{\overline{u}(t_k;i)\}} \cdot [u(t_k,i) - \overline{u}(t_k,i)] + v_y(t_j) \tag{25}$$

TABLE 1. The partial derivatives necessary to estimate the control $u(t)$

J	k	$j-1$	Partial Derivatives
1	0	0	$\left.\frac{\partial \hat{y}(t_1)}{\partial u(t_0)}\right\|_{\overline{u}(t_0,i)}$

In order for the partial vector derivative presented in Table 1 to be computed correctly it is necessary to establish a priori the type of numerical integrator in which the neural derivative function of the dynamic system to be studied will be introduced. In the present case the Adams-Bashforth multi-step integrator of order four will be adopted, given by:

$$\begin{aligned} y(t_1) = y(t_0) + \frac{h}{24} \cdot \Big\{ & 55 \cdot \hat{f}[y(t_0), u(t_0)] - 59 \cdot \hat{f}[y(t_{-1}), u(t_{-1})] \\ & + 37 \cdot \hat{f}[y(t_{-2}), u(t_{-2})] - 9 \cdot \hat{f}[y(t_{-3}), u(t_{-3})] \Big\} \end{aligned} \tag{26}$$

Thus, the calculation of the derivative $\left.\frac{\partial \hat{y}(t_1)}{\partial u(t_0)}\right|_{\overline{u}(t_0,i)}$, for the particular case of the adopted integrator, takes the following form:

$$\left.\frac{\partial \hat{y}(t_1)}{\partial u(t_0)}\right|_{\overline{u}(t_0,i)} = \frac{55}{24} \cdot h \cdot \left.\frac{\partial \hat{f}[y(t_0), u(t_0)]}{\partial u(t_0)}\right|_{\overline{u}(t_0,i)} \tag{27}$$

The above equation can be computed directly through backpropagation algorithm. For further details on the calculation of these derivatives see Appendix A.

3) Mount the matrix $H^u(t,i)$ and the vector $\overline{Z}^u(t,i)$ present in the equation $\alpha(i) \cdot \overline{Z}^u(t,i) = H^u(t,i) \cdot [U(t,i) - \overline{U}(t,i)] + V_y(t)$. The matrix $H^u(t,i)$ must be constructed from Table 1. To simplify the notation being used, it is convenient to adopt the following definition:

$$A_{k,j} = \left.\frac{\partial \hat{y}(t_k)}{\partial u(t_j)}\right|_{\overline{u}(t_j,i)} = \begin{bmatrix} \frac{\partial \hat{y}_1(t_k)}{\partial u_1(t_j)} & \frac{\partial \hat{y}_1(t_k)}{\partial u_2(t_j)} & \cdots & \frac{\partial \hat{y}_1(t_k)}{\partial u_{n_u}(t_j)} \\ \frac{\partial \hat{y}_2(t_k)}{\partial u_1(t_j)} & \frac{\partial \hat{y}_2(t_k)}{\partial u_2(t_j)} & \cdots & \frac{\partial \hat{y}_2(t_k)}{\partial u_{n_u}(t_j)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_{n_y}(t_k)}{\partial u_1(t_j)} & \frac{\partial \hat{y}_{n_y}(t_k)}{\partial u_2(t_j)} & \cdots & \frac{\partial \hat{y}_{n_y}(t_k)}{\partial u_{n_u}(t_j)} \end{bmatrix} \tag{28}$$

Assembling matrix $H^u(t,i)$ would be a rather difficult task if the values of n, n_y and n_u were large, for example, if $n = 5$ and $n_y = n_u = 3$. In this case, for illustrative purposes only, the following would have occurred:

$$H^u(t,i) = \begin{bmatrix} A_{1,0} & 0 & 0 & 0 & 0 \\ A_{2,0} & A_{2,1} & 0 & 0 & 0 \\ A_{3,0} & A_{3,1} & A_{3,2} & 0 & 0 \\ A_{4,0} & A_{4,1} & A_{4,2} & A_{4,3} & 0 \\ A_{5,0} & A_{5,1} & A_{5,2} & A_{5,3} & A_{5,4} \end{bmatrix} \tag{29}$$

where each generic element $A_{k,j}$ present in the previous matrix would be another matrix of dimension $n_y \times n_u$ as indicated by Equation (28). However, since $n = 1$ then the matrix $H^u(t,i)$ reduces to:

$$H^u(t,i) = A_{1,0} = \left.\frac{\partial \hat{y}(t_1)}{\partial u(t_0)}\right|_{\overline{u}(t_0,i)} = \left[\left.\frac{\partial \hat{y}_j(t_1)}{\partial u_k(t_0)}\right|_{\overline{u}(t_0,i)}\right]_{n_y \times n_u} \tag{30}$$

Thus, by expanding the expression $\alpha(i) \cdot \overline{Z}^u(t,i) = H^u(t,i) \cdot [U(t,i) - \overline{U}(t,i)] + V_y(t)$ one has:

$$\underbrace{\begin{bmatrix} y_{r_1}(t_1) - \overline{y}_1(t_1,i) \\ y_{r_2}(t_1) - \overline{y}_2(t_1,i) \\ \vdots \\ y_{r_{n_y}}(t_1) - \overline{y}_{n_y}(t_1,i) \end{bmatrix}}_{\overline{Z}^u(t,i)} \cdot \alpha(i) = \underbrace{\begin{bmatrix} \frac{\partial \hat{y}_1(t_1)}{\partial u_1(t_0)} & \frac{\partial \hat{y}_1(t_1)}{\partial u_2(t_0)} & \cdots & \frac{\partial \hat{y}_1(t_1)}{\partial u_{n_u}(t_0)} \\ \frac{\partial \hat{y}_2(t_1)}{\partial u_1(t_0)} & \frac{\partial \hat{y}_2(t_1)}{\partial u_2(t_0)} & \cdots & \frac{\partial \hat{y}_2(t_1)}{\partial u_{n_u}(t_0)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_{n_y}(t_1)}{\partial u_1(t_0)} & \frac{\partial \hat{y}_{n_y}(t_1)}{\partial u_2(t_0)} & \cdots & \frac{\partial \hat{y}_{n_y}(t_1)}{\partial u_{n_u}(t_0)} \end{bmatrix}}_{H^u(t,i)}\overline{u}(t_0,i)$$

$$\cdot \underbrace{\begin{bmatrix} u_1(t_0,i) - \overline{u}_1(t_0,i) \\ u_2(t_0,i) - \overline{u}_2(t_0,i) \\ \vdots \\ u_{n_u}(t_0,i) - \overline{u}_{n_u}(t_0,i) \end{bmatrix}}_{U(t,i)-\overline{U}(t,i)} + \underbrace{\begin{bmatrix} v_1(t_1) \\ v_2(t_1) \\ \vdots \\ v_{n_y}(t_1) \end{bmatrix}}_{V_y(t)} \tag{31}$$

where $\overline{U}(t,i)$ is the a priori information vector of the control variables adopted in step 1; $U(t,i)$ is the vector that should be estimated; $H^u(t,i)$ is the partial derivative matrix obtained in step 2 through backpropagation and assembled in step 3; $0 < \alpha(i) \leq 1$ is the value that must be adopted empirically to guarantee the linearization hypothesis; and $V_y(t)$ is the mean zero noise vector and covariance matrix $r_y(t)$.

4) Estimate $\hat{U}(t,i)$ through the following iterative expressions of the Kalman filtering (see [10]):

$$\begin{aligned} k(t,i) &= R_u(t) \cdot H^{u^T}(t,i) \cdot \left[H^u(t,i) \cdot R_u(t) \cdot H^{u^T}(t,i) + R_y(t)\right]^{-1} \\ &\equiv \left[R_u^{-1}(t) + H^{u^T}(t,i) \cdot R_y^{-1}(t) \cdot H^u(t,i)\right]^{-1} \cdot H^{u^T}(t,i) \cdot R_y^{-1}(t) \end{aligned} \tag{32a}$$

$$\begin{aligned} \hat{U}(t,i) = \overline{U}(t,i) &+ \alpha(i) \cdot \left[\hat{U}(t_{-1}) - \overline{U}(t,i)\right] + k(t,i) \cdot \alpha(i) \\ &\cdot \left[\overline{Z}^u(t,i) - H^u(t,i) \cdot \left[\hat{U}(t_{-1}) - \overline{U}(t,i)\right]\right] \end{aligned} \tag{32b}$$

where $R_u(t)$ and $R_y(t)$ are the covariance matrices, respectively, of $V_u(t)$ and $R_y(t)$; $\hat{U}(t_{-1})$ is the previous estimate of the control variables within an acceptable precision. Note that the processing of Equations (32a) and (32b) can be done both in batches and recursively.

5) Implement the variable i of $i = i + 1$, and do $\overline{U}(t, i+1) = \hat{U}(t, i)$ and repeat the steps (2), (3) and (4) until that $y(t)$ is sufficiently close to $y_r(t)$ within a desired error, in general, $3 \cdot \sigma$ of v_y. One can use the criterion of the mean square error to determine the last iteration for the present estimation. When this occurs, then do:

$$\hat{R}_u(t, I) = [I_u - k(t, I) \cdot H^u(t, I)] \cdot R_u(t) \tag{33}$$

where $\hat{R}_u(t, I)$ is the estimation of the covariance matrix of $\left[\hat{U}(t, I) - U(t)\right]$.

6) When the current estimation converges to the desired value $y_r(t)$, advance only one instant of time forward, even if the forecast horizon adopted n is greater than one, and repeat all previous steps for the new estimation. Do not forget that now $\hat{U}(t_{-1}) = \hat{U}(t, I)$ and $\overline{U}(t, 1) = \hat{U}(t, I)$.

5. **Numerical Results.** The practical results of the methodology developed in this paper will be presented for the problem of orbit transfer between the planets Earth and Mars, as shown in Figure 5. Figures 6 and 7 show the numerical results of this orbiting transfer problem of a rocket when the integration step is varied. The state variables of this problem are: the mass of the rocket m, the orbit radius r, the radial velocity w and the transverse velocity v. The only control variable is the guided thrust angle θ. The state equations (see [26]) of the dynamics of Earth-Mars orbit transfer of mass-rocket m are:

$$\dot{m} = -0.0749 \tag{34a}$$

$$\dot{r} = w \tag{34b}$$

$$\dot{w} = \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T \cdot \sin\theta}{m} \tag{34c}$$

$$\dot{v} = \frac{-w \cdot v}{r} + \frac{T \cdot \cos\theta}{m} \tag{34d}$$

The normalized constants of this problem are: $\mu = 1.0$ (gravitational constant), $T = 0.1405$ (rocket thrust), $t_0 = 0$ (initial moment) and $t_f = 3.3$ (final moment), where each unit of time is equivalent to 58.2 days. The input/output training patterns generated

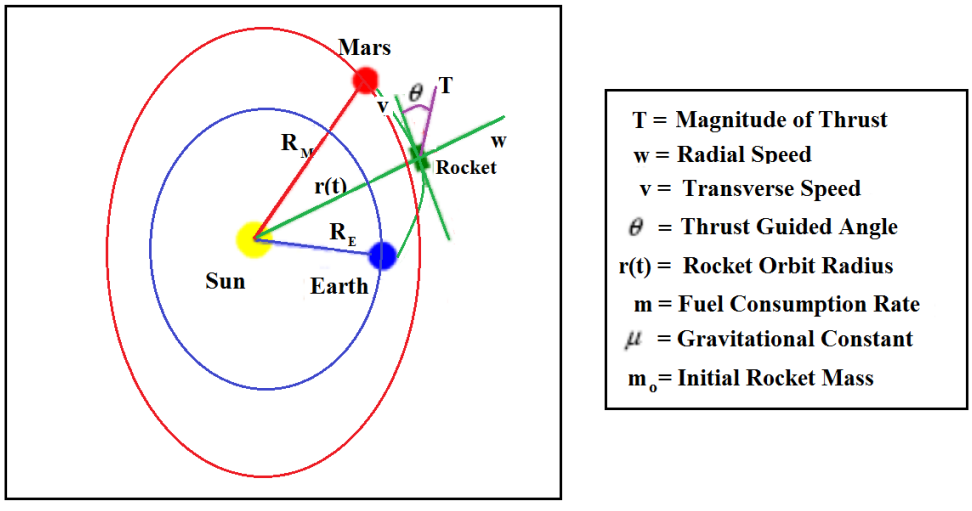

FIGURE 5. Illustrative diagram of Earth-Mars orbit transfer

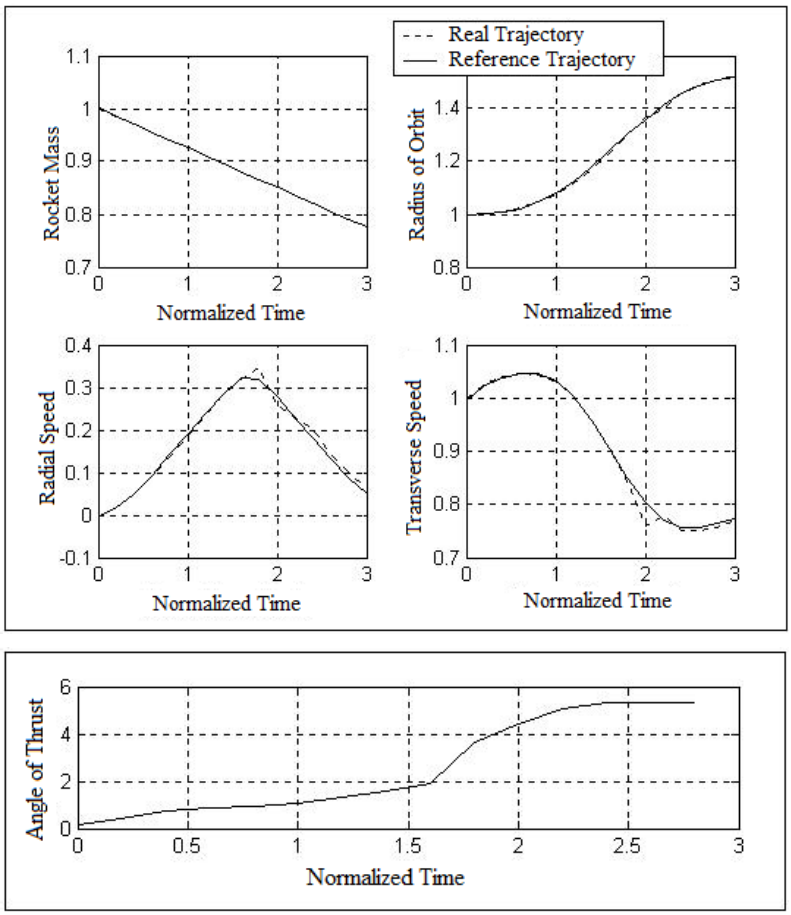

FIGURE 6. Neural predictive control structure with integration step of $\Delta t = 0.2$

to train the fourth-order Adams-Bashforth neural network were generated numerically from the classical Runge-Kutta 4-5 numerical integrator on the non-linear equations of the dynamics from (34a) to (34d).

In Figure 6 we have the result of neural predictive control, where the system starts with negligible error in relation to the beginning of the reference trajectory. However, since it has an integration and estimation step that are equal to each other and with a value that is relatively high and equal to 0.2, the final results of the simulation are not very accurate.

The numerical results of Figure 7 show that, for a smaller integration step than that used in Figure 6, it is possible to better track the rocket reference trajectory. The lower part of Figures 6 and 7 also shows the estimated control. Note that in both cases the same fourth-order Adams-Bashforth neural network was used. The only thing that varied was the integration step. Note that if the NARMAX methodology had been used one would have to train two neural networks in order to vary the integration step from 0.2 to 0.01.

6. **Conclusions.** In this work a new approach to obtaining the discrete models of dynamic systems, where an internal model is needed, was presented and tested, in a predictive control scheme. The possibility of using universal numerical integrators of differential equation systems as internal models was considered. It was shown that the structure of these numerical integrators can be exploited to obtain discrete neural models, where the

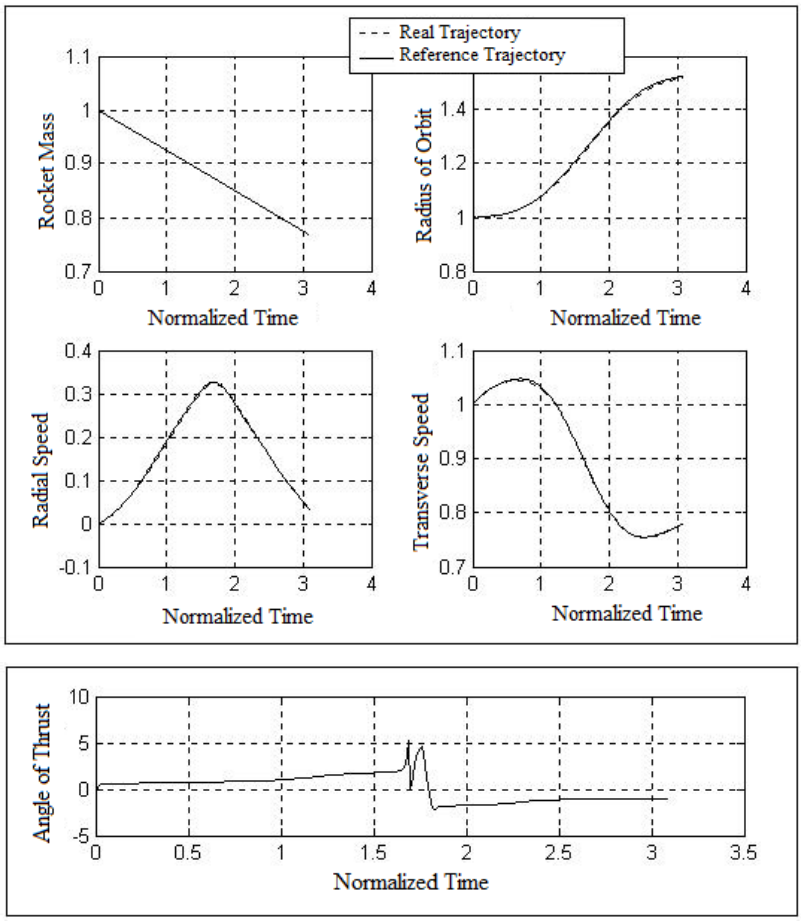

FIGURE 7. Neural predictive control structure with integration step of $\Delta t = 0.01$

neural network has only to learn and approximate instantaneous derivatives functions that are simply of static algebraic nature.

Conclusions, previously perceived by the proponents of the approach, could be confirmed:

a) it is a simpler task to train a neural network with feedforward architecture to learn an algebraic and static function (in this case, the instantaneous derivatives functions of the ordinary differential equations model of the dynamic system) than to train it to learn the discrete dynamic model through the NARMAX methodology;

b) the architecture of the resulting neural network is simpler as regard to the number of layers and number of neurons, since it does not have to learn the law of dynamics, but only the function of instantaneous derivatives;

c) the use of the universal numerical integrator for systems of ordinary differential equations, as an approximate model of discrete time, does not destroy the parallel processing characteristic of the neural network, since the numerical integration algorithm only involves calculations and evaluations of linear combinations of the neural network trained;

d) the flexibility of being able to vary the size of the step of discretization and the order of the universal numerical integrator, can be used to control the desired precision for the system response output, since the neural network is trained with a tolerance compatible with that which is desired in the universal numerical integrator;

e) when the response times are not too small, and the mathematical model of ordinary differential equations evaluated is reasonably good, then the structure of the universal numerical integrator can be directly used as a discrete internal model;

f) even in situations where a theoretical mathematical model cannot be evaluated and there are only pairs of dynamic input/output information of the system, experimentally obtained, the structure of the universal numerical integrator with a feedforward network in place of the theoretical instantaneous derivative functions can be trained to obtain a discrete internal model in control schemes;

g) finally, it is important to consider that the use of a neural network in a discrete model of the dynamic system will naturally allow the implementation of adaptive control schemes due to the learning capacity of neural networks.

As pointed out, there are three distinct ways of solving the problem of nonlinear programming required by the predictive control structure:

a) to use only the neural network (or any other universal approximation of functions) to approximate the dynamics of the system using the criterion of delayed inputs (NARMAX methodology);

b) to use only the numerical integrator as discretized model of the system dynamics; or

c) to use together the numerical integrator and a neural network that will play the role of representing the dynamic system functions of instantaneous derivatives.

Using only the neural network leads to a more "robust" (more inputs and more neurons) network and therefore more difficult to be trained, besides not allowing the variation of the integration step size.

Using only the numerical integrator structure, with the theoretical instantaneous derivative function, makes it difficult to adapt the theoretical model of the dynamic system to the real plant.

Using the numerical integrator structure with the neural network to approximate the instantaneous derivative function allows the construction of a less robust network and, therefore, an easier training, in addition to being able to design a dynamic model with variable integration step and which can be adapted in real time to the original system of the plant.

The major disadvantage of the former method is that the computation of the partial derivatives becomes more complex and complicated, and for each type of integrator to be used there will be a different expression for the partial derivatives and, in general, these expressions will be so much more complex and difficult to obtain as the order of the integrator becomes larger.

The use of the Adams-Bashforth multi-step neural integrator as an internal model, in the predictive control scheme based on the Kalman filtering algorithm, applied to the Earth/Mars transfer problem, resulted in the following results:

a) the stochastic interpretation of the errors gave more realism to the treatment of the problem and facilitated the adjustment of the weight matrices in the functional of the predictive control problem;

b) the Kalman algorithm for the calculation of the control actions performed similarly to the corresponding Kalman algorithm for neural network training, as expected, since they are completely similar algorithms, applied to solving numerically equivalent problems of parameter estimation;

c) based on the previous item, the Kalman filtering, with or without parallel processing, can be used both to train the neural network and to find the optimal control policy;

d) the use of only one step forward as a control horizon was enough; this feature, coupled with the efficiency and performance of Kalman's parallel processing algorithms

combined with today's on-board processing capability, ensures the feasibility of real-time adaptive applications in aerospace control problems.

Acknowledgments. Initially, the authors would like to thank for the excellent technical review and good suggestions made by the reviewers of this paper. We would also like to thank the Technological Institute of Aeronautics (ITA), Casimiro Montenegro Filho Foundation (FCMF) and the 2RP Net Enterprise staffs for sponsoring this research.

REFERENCES

[1] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, vol.2, no.5, pp.359-366, 1989.
[2] N. E. Cotter, The Stone-Weierstrass and its application to neural networks, *IEEE Trans. Neural Networks*, vol.1, no.4, pp.290-295, 1990.
[3] J.-S. R. Jang, C.-T. Sun and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Inc., 1997.
[4] K. J. Hunt, D. Sbarbaro, R. Zbikowski and P. J. Gawthrop, Neural networks for control systems – A survey, *Automatica*, vol.28, no.6, pp.1083-1112, 1992.
[5] D. W. Clarke, C. Mohtadi and P. S. Tuffs, Generalized predictive control – Part I. The basic algorithm, *Automatica*, vol.23, no.2, pp.137-148, 1987.
[6] D. W. Clarke, C. Mohtadi and P. S. Tuffs, Generalized predictive control – Part II. Extensions and interpretations, *Automatica*, vol.23, no.2, pp.149-160, 1987.
[7] G. P. Liu, V. Kadirkamanathan and S. A. Billings, Predictive control for non-linear systems using neural networks, *Int. J. Control*, vol.71, no.6, pp.1119-1132, 1998.
[8] M. A. Botto and J. S. da Costa, A comparison of nonlinear predictive control techniques using neural network models, *Journal of Systems Architecture*, vol.44, no.8, pp.597-616, 1998.
[9] P. H. Sorense, M. Norgaard, O. Ravn and N. K. Poulsen, Implementation of neural network based non-linear predictive control, *Neurocomputing*, vol.28, nos.1-3, pp.37-51, 1999.
[10] A. R. Neto, Design of a Kalman filtering based neural predictive control method, *XIII Congresso Brasileiro de Automática (CBA)*, Florianópolis/SC, Brazil, pp.2130-2134, 2000.
[11] A. R. Neto, Dynamic systems numerical integrators in neural control schemes, *V Congresso Brasileirode Redes Neurais*, Rio de Janeiro-RJ, Brazil, pp.85-88, 2001.
[12] P. M. Tasinaffo, *Estruturas de Integração Neural Feedforward Testadas em Problemas de Controle Preditivo*, Ph.D. Thesis, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos/SP, Brazil, 2003.
[13] P. Henrici, *Elements of Numerical Analysis*, John Wiley & Sons, New York, 1964.
[14] J. D. Lambert, *Computational Methods in Ordinary Differential Equations*, John Wiley & Sons, New York, 1973.
[15] M. Vidyasagar, *Nonlinear Systems Analysis*, Networks Series, Electrical Engineering Series, Prentice-Hall, NJ, 1978.
[16] M. Braun, *Differential Equations and Their Applications: An Introduction to Applied Mathematics*, 3rd Edition, Applied Mathematical Sciences, Springer-Verlag, New York, 1983.
[17] K. R. Rao, *A Review on Numerical Methods for Initial Value Problems*, São José dos Campos/SP, Brazil, 1984.
[18] A. R. Neto and K. R. Rao, A stochastic approach to global error estimation in ODE multistep numerical integration, *Journal of Computational and Applied Mathematics*, vol.30, no.3, pp.257-281, 1990.
[19] Y.-J. Wang and C.-T. Lin, Runge-Kutta neural network for identification of dynamical systems in high accuracy, *IEEE Trans. Neural Networks*, vol.9, no.2, pp.294-307, 1998.
[20] R. P. Melo and P. M. Tasinaffo, Uma metodologia de modelagem empírica utilizando o integrador neural de múltiplos passos do tipo Adams-Bashforth, *SBA. Sociedade Brasileira de Automática*, vol.21, no.5, pp.487-509, 2010.
[21] R. P. Melo, *Metodologia de Modelagem Empírica Utilizando Integradores Neurais de Múltiplos-passos Aplicado a Sistemas Dinâmicos Não-lineares*, Master Thesis, Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos/SP, Brazil, 2008.
[22] P. M. Tasinaffo and A. R. Neto, Predictive control with mean derivative based neural Euler integrator dynamic model, *Sociedade Brasileira de Automática (SBA)*, vol.18, no.1, pp.94-105, 2006.

[23] P. M. Tasinaffo, R. dos Santos Guimarães, L. A. V. Dias, V. Strafacci Júnior and F. R. M. Cardoso, Discrete and exact general solution for nonlinear autonomous ordinary differential equations, *International Journal of Innovative Computing, Information and Control*, vol.12, no.5, pp.1703-1719, 2016.

[24] M. O. de Figueiredo, P. M. Tasinaffo and L. A. V. Dias, Modeling autonomous nonlinear dynamic systems using mean derivatives, fuzzy logic and genetic algorithms, *International Journal of Innovative Computing, Information and Control*, vol.12, no.5, pp.1721-1743, 2016.

[25] P. M. Tasinaffo, R. dos Santos Guimarães, L. A. V. Dias and V. Strafacci Júnior, Mean derivatives methodology by using Euler integrator improved to allow the variation in the size of integration step, *International Journal of Innovative Computing, Information and Control*, vol.12, no.6, pp.1881-1891, 2016.

[26] A. P. Sage, *Optimum Systems Control*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1968.

[27] V. Carrara, *Redes Neurais Aplicadas ao Controle de Atitude de Satélites com Geometria Variável*, Ph.D. Thesis, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos/SP, Brazil, 1997.

Appendix A: The Calculation of Backpropagation Algorithm. Figure 8 shows a simplified graphical scheme of a generic layer in a feedforward network (see [27]). The formal mathematical representation for the output vector y^k of the layer k will be given by the following expression:

$$y_i^k = f^k\left(\sum_{j=1}^{n_{k-1}} w_{ij}^k \cdot y_j^{k-1} - b_i^k\right) \quad \text{for } i = 1, 2, \ldots, n_k \tag{35}$$

The analytical processing given by Equation (35) can be placed in the matrix form, demonstrating the parallelism characteristic presented by MLP networks. Thus, W^k being the weight matrix of the layer k ($k = 1, 2, \ldots, L$), the output vector of this layer will be:

$$y^k = f^k\left(W^k \cdot y^{k-1}\right) = f^k\left(\overline{y}^k\right) \tag{36a}$$

where

$$y^k = \left[f^k\left(\overline{y}_1^k\right) \cdots f^k\left(\overline{y}_{n_k}^k\right) - 1\right]^T \tag{36b}$$

$$W^k = \begin{bmatrix} w_{11}^k & \cdots & w_{1n_{k-1}}^k & b_1^k \\ w_{21}^k & \cdots & w_{2n_{k-1}}^k & b_2^k \\ \vdots & \ddots & \vdots & \vdots \\ w_{n_k1}^k & \cdots & w_{n_kn_{k-1}}^k & b_{n_k}^k \end{bmatrix}_{n_k, n_{k-1}+1} \tag{36c}$$

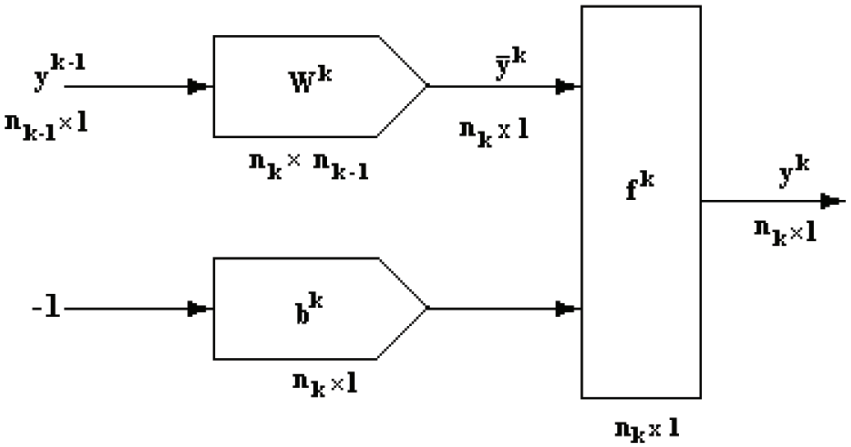

FIGURE 8. Inputs and outputs of a generic layer k of an MLP network (source: see [27])

It should be noted that in Equation (36c) the line vector b^k was added to the weight matrix W^k and because this it is necessary to add the negative unit value in the last line of vector y^k represented by Equation (36b).

Thus, the calculation of backpropagation can be determined (see [27]) as follows:

$$\frac{\partial y^j}{\partial \overline{y}^k} = \frac{\partial y^j}{\partial \overline{y}^{k+1}} \cdot W^{k+1} \cdot I_{f'\left(\overline{y}^k\right)} \quad \text{for } k = L-1, L-2, \ldots, 1 \tag{37a}$$

where

$$\frac{\partial y^L}{\partial \overline{y}^L} = I_{f'\left(\overline{y}^L\right)} \quad \text{for } j = k = L \text{ (the output layer of MLP network)} \tag{37b}$$

$$I_{f'\left(\overline{y}^L\right)} = \begin{bmatrix} f'\left(\overline{y}_1^L\right) & 0 & 0 & 0 \\ 0 & f'\left(\overline{y}_2^L\right) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & f'\left(\overline{y}_{n_k}^L\right) \end{bmatrix} \tag{37c}$$

$$\left.\frac{\partial \hat{f}[y(t_0), u(t_0)]}{\partial u(t_0)}\right|_{\overline{u}(t_0,i)} = A_u \tag{37d}$$

$$\frac{\partial y^L}{\partial \overline{y}^1} = \left[\frac{\partial \hat{f}[y(t_0), y(t_0)]}{\partial y(t_0)} : \frac{\partial \hat{f}[y(t_0), u(t_0)]}{\partial u(t_0)}\right] = [A_y : A_u] \tag{37e}$$

$$\frac{\partial y^L}{\partial \overline{y}^1} = \frac{\partial \hat{y}^L}{\partial \overline{y}^1} = \underbrace{\begin{bmatrix} \frac{\partial \hat{y}_1^L}{\partial \overline{y}_1^1} & \frac{\partial \hat{y}_1^L}{\partial \overline{y}_2^1} & \cdots & \frac{\partial \hat{y}_1^L}{\partial \overline{y}_{n_y}^1} \\ \frac{\partial \hat{y}_2^L}{\partial \overline{y}_1^1} & \frac{\partial \hat{y}_2^L}{\partial \overline{y}_2^1} & \cdots & \frac{\partial \hat{y}_2^L}{\partial \overline{y}_{n_y}^1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_{n_y}^L}{\partial \overline{y}_1^1} & \frac{\partial \hat{y}_{n_y}^L}{\partial \overline{y}_2^1} & \cdots & \frac{\partial \hat{y}_{n_y}^L}{\partial \overline{y}_{n_y}^1} \end{bmatrix}}_{A_y \equiv \frac{\partial \hat{f}[y(t_0), u(t_0)]}{\partial y(t_0)}} \underbrace{\begin{bmatrix} \frac{\partial \hat{y}_1^L}{\partial \overline{y}_{n_y+1}^1} & \frac{\partial \hat{y}_1^L}{\partial \overline{y}_{n_y+2}^1} & \cdots & \frac{\partial \hat{y}_1^L}{\partial \overline{y}_{n_y+n_u}^1} \\ \frac{\partial \hat{y}_2^L}{\partial \overline{y}_{n_y+1}^1} & \frac{\partial \hat{y}_2^L}{\partial \overline{y}_{n_y+2}^1} & \cdots & \frac{\partial \hat{y}_2^L}{\partial \overline{y}_{n_y+n_u}^1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_{n_y}^L}{\partial \overline{y}_{n_y+1}^1} & \frac{\partial \hat{y}_{n_y}^L}{\partial \overline{y}_{n_y+2}^1} & \cdots & \frac{\partial \hat{y}_{n_y}^L}{\partial \overline{y}_{n_y+n_u}^1} \end{bmatrix}}_{A_u \equiv \frac{\partial \hat{f}[y(t_0), u(t_0)]}{\partial u(t_0)}} \tag{37f}$$

Equations (37a) to (37f) are used to calculate the partial derivative matrix present in Equation (27) and refer to the computational algorithm of the predictive neural control structure detailed in Section 4.2.