Direct Versus Indirect Neural Control Based on Radial Basis Function Networks

Alex Alexandridis[†], Marios Stogiannos[†], Andronikos Loukidis[†], Konstantinos Ninos[†], Evangelos Zervas[†], Haralambos Sarimveis[*]

[†]Department of Electronic Engineering, Technological Educational Institute of Athens, Greece
alexx@teiath.gr

[*]School of Chemical Engineering, National Technical University of Athens, Greece

***Abstract*—This work presents a comparison between direct and indirect neural control methods based on the radial basis function (RBF) architecture. As far as direct control schemes are concerned, a novel direct inverse neural RBF controller taking into account the applicability domain criterion (INCAD) is utilized. A model predictive control (MPC) formulation based on RBF networks is tested as an example of indirect method. The performances of the two control schemes are evaluated and compared on a highly nonlinear control problem, namely control of a continuous stirred tank reactor (CSTR) with multiple stable and unstable steady states. Results show that the INCAD controller is able to provide satisfactory performance, while performing almost instant calculation of the control actions. MPC on the other hand, outperforms the INCAD in terms of speed of responses, due to the built-in optimization capability; however, the lengthy procedure of solving online the optimization problem impedes the practical use of MPC on systems with fast dynamics.**

Keywords—Direct control; indirect control; model predictive control ; neurocontrol; radial basis function

I. INTRODUCTION

Artificial neural networks (NNs) [1] have been used extensively during the last decades for designing novel automatic control schemes [2, 3]. The success of NN-based controllers is mainly due to their ability to learn from experimental data using specially designed efficient online or offline training methods, a fact which makes them particularly useful in cases where conventional methodologies fail to develop appropriate dynamic models of the system and/or compute the control law. Other useful properties of NNs include tolerating faults and uncertainties, acting as universal approximators, and performing massive parallel processing. Among the different NN architectures used for designing control systems, radial basis function (RBF) networks [4] present some important advantages including better approximation capabilities, simpler network structures, and faster learning algorithms.

There are two basic design approaches for formulating NN-based control strategies: *Indirect design*, where the NN serves as the dynamic model of the system, predicting the output or the complete state vector and *direct design* where the NN approximates the inverse system dynamics and acts as a controller. The indirect design approach is usually coupled with the Model Predictive Control (MPC) methodology, where NN methodologies are applied on historical dynamic input-output data to construct the nonlinear dynamic predictive model of the system [5, 6]. An optimization procedure is then applied to find the optimum sequence of moves that will drive the system to the desired conditions. MPC techniques can handle multiple input–multiple output (MIMO) systems, and take into account input/output/state constraints and system/model mismatches.

Despite their advantages, indirect design methodologies based on the MPC concept share an important drawback: the nonlinear optimization problem must be solved in real time, before the next sample is collected from the system; nevertheless, the available time between two subsequent time instants may not be sufficient even for satisfying the requirements of suboptimal MPC. This limitation prevents the application of indirect design methodologies to systems with fast dynamics. Direct design techniques on the other hand, completely avoid the solution of the optimization problem, using the NN as an explicit control law. This means that at each discrete time instant, the NN directly calculates the values of the manipulated variables that are used as inputs to the system [7, 8]. Past values of input and output (or state) variables along with the desired set-points constitute the input vector to the inverse NN model. The efficiency of direct design methodologies stems from the fact that their implementation requires only the evaluation of a nonlinear function at each time instant, as opposed to the lengthy solution of the nonlinear optimization problem.

In this work we select a novel direct neural control scheme with the additional ability to avoid extrapolation based on the applicability domain (AD) concept [9] and test it against a neural indirect MPC controller. Both controllers, which are based on the RBF architecture, are applied to the control of a continuous stirred chemical reactor (CSTR), exhibiting multiple steady states.

The work of M. Stogiannos was supported by the Greek State Scholarship Foundation (IKY), through the IKY Fellowships of Excellence for Postgraduate Studies in Greece - Siemens Program.

The rest of this paper is organized as follows: In the next section we give a brief introduction to RBF networks and the FM algorithm. In Section 3 we describe the development of RBF-based neural controllers, including the direct and indirect approaches. Section 4 presents a comparison of the two different approaches when applied to a nonlinear control problem. Finally, the paper ends with the concluding section.

II. RBF Networks and the FM Algorithm

RBF networks form a special neural network architecture that consists of three layers (Fig. 1).The RBF network training procedure corresponds to determining the hidden node basis function parameters and the output-layer weights. Standard approaches in RBF network training, decompose this problem in two steps: In the first step, the hidden layer basis function parameters are obtained using an unsupervised clustering technique, like the *k*-means algorithm [10]. The latter postulates an arbitrary number of RBF centers and then calculates their locations, the final selection being made through a tedious trial-and-error procedure. An alternative to this time-consuming approach was given by the fuzzy means (FM) algorithm [4], which has the ability to calculate in one step the number and locations of the hidden node centers. The FM algorithm has found many successful applications in fields like medical diagnosis [11], soft-sensor development [12], etc.

The FM algorithm, is based on a fuzzy partition of the input space, where the domain of each input variable is partitioned into a number of fuzzy sets. Combining *N* one-dimensional fuzzy sets, where *N* is the number of input dimensions, one can generate a multi-dimensional fuzzy subspace. These fuzzy subspaces form a grid where each node is candidate for becoming an RBF center. The objective of the FM algorithm is to assemble the RBF network hidden layer by selecting only a small subset of the fuzzy subspaces. This selection is based on a hyper-circle placed around each fuzzy subspace center. The hyper-circle marks a boundary between input vectors that receive non-zero or zero membership degrees to each particular fuzzy subspace. Having defined the membership function, the algorithm proceeds with finding the subset of fuzzy subspaces that assign a nonzero multidimensional degree to all input training vectors. This is accomplished using a non-iterative algorithm which requires only one pass of the input data, thus rendering the center calculation procedure extremely fast, even in the presence of a large database of input examples. More details about the FM algorithm can be found in [4].

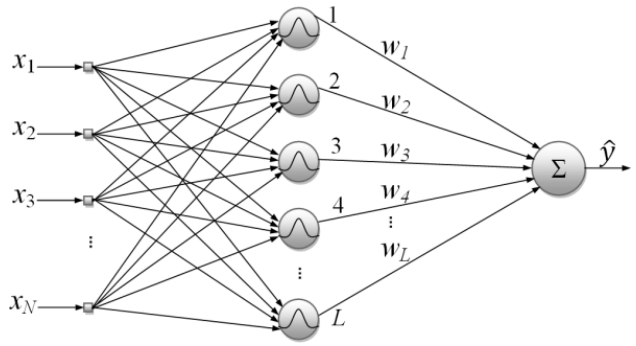

Fig. 1. Typical structure of an RBF network

The second step in RBF training involves determination of the output-layer weights; this step is usually implemented by linear least squares regression, which guarantees a global minimum as far as the weights are concerned.

III. Neural controller design based on RBF networks

A. Indirect design using MPC

Let us assume that the system under control is a Single Input-Single Output (SISO) system with *N* state variables. A discrete dynamic model of the system correlates the next value of the output (controlled) variable with the current values of the state variables and the input (manipulated) variable:

$$y(k+1) = f\left(\mathbf{x}(k), v(k)\right) \tag{1}$$

where y is the controlled variable, $\mathbf{x}$ is the state vector and v is the manipulated variable. We can assume, without loss of generality, that f is a nonlinear function.

MPC controllers typically make use of a direct dynamic model of the system, thus approximating (1). Under the assumptions that all state variables are measurable and a sufficient number of training examples is available, the FM algorithm can be used to approximate the unknown function f, providing a model able to predict the next value of the output variable based on previous values of the states and the input:

$$y(k+1) = \mathrm{RBF}\left(\mathbf{x}(k), v(k)\right) \tag{2}$$

where RBF is the nonlinear function implemented by the RBF network, calculated through (2). Notice that this model cannot by itself be used to control the plant, as it is trained only to identify the unknown system and predict the result of candidate control actions. In order to make use of (2), at each discrete time step an optimization problem is formulated, where the objective is to minimize the difference between the set point and the predictions of the model. The solution to this problem is the optimum sequence of control moves that drives the controlled variable to the set point value.

In a typical MPC implementation, the objective function at each discrete time instant k is a combination of two targets: a) minimization of the distances between the predicted output values and the set point and b) minimization of the control moves:

$$\min_{v(k),\ldots,v(k+h_c)} \sum_{i=1}^{h_p}\left(\Theta_t\left(\hat{y}(k+i)-\omega(k)\right)\right)^2 + \sum_{i=0}^{h_c}\left(\Omega_i \Delta v(k+i)\right)^2 \tag{3}$$

where Δv is the difference between two subsequent control moves; $\boldsymbol{\Theta}$ and $\boldsymbol{\Omega}$ are the error and move suppression weights; h_c and h_p are the control and prediction horizons, respectively, $\hat{y}$ is the model prediction for the controlled variable, which is assumed to be one of the state variables, and ω is the set point value.

The minimization problem is solved subject to a number of constraints:

$$\hat{y}(k+i) = \mathrm{RBF}\left(\mathbf{x}(k+i-1), v(k+i-1)\right) + E(k+i), \quad 1 \le i \le h_p \tag{4}$$

$$E(k+i) = y(k) - \mathrm{RBF}\left(\mathbf{x}(k-1), v(k-1)\right), \ 1 \le i \le h_p \tag{5}$$

$$v_{\min} \le v(k+i) \le v_{\max}, \ 1 \le i \le h_c \tag{6}$$

$$\Delta v(k+i) = 0, \ h_c + 1 \le i \le h_p \tag{7}$$

where $E(k)$ is the current error between the actual output measurement and the model prediction. Eqs. (4)-(5) denote that the modeling error measured at time point k is assumed to be the same over the entire prediction horizon. Notice that multiple calls of the RBF model (2) are needed to calculate the future values of the controlled variable y, throughout the prediction horizon. Eq. (6) poses upper and lower bounds on the values of the manipulated variables, while (7) denotes that no control moves are allowed after the end of the control horizon. The set of constraints can be augmented by imposing upper and lower bounds on the system output or the state vector. Figure 2 shows schematically how the RBF model and the MPC methodology are integrated.

B. Direct design using inverse RBF models

The concept of a direct controller is simpler compared to the indirect MPC scheme described in the previous subsection. Direct design implementations are based on a discrete inverse dynamic model of the system under control that can be formulated as follows:

$$v(k) = g\left(\mathbf{x}(k), y(k+1)\right) \tag{8}$$

where g is the inverse dynamic function of the system. Substituting the next value of the controlled variable $y(k+1)$ in (8) with the current value of the setpoint $\omega(k)$, essentially provides a control law which can be used for calculating which value of the manipulated variable should be applied at any discrete time instant k, in order to drive the system from its current state, to the setpoint:

$$v(k) = g\left(\mathbf{x}(k), \omega(k)\right) \tag{9}$$

Under the same assumptions of measurable states and adequate number of training examples, the FM algorithm can be used to approximate the unknown function g. The RBF approximator of the inverse dynamics can then act as a controller, receiving at each discrete time instant the desired set-point value and the values of the state variables and producing the control command:

$$v(k) = \mathrm{RBF}\left(\mathbf{x}(k), \omega(k)\right) \tag{10}$$

The operation of such a controller is depicted in Fig. 3.

It is well known that neural network models – as well as other black-box modeling techniques – have a limited ability for generalization. This comes as a direct result of the inadequacy of such models to provide reliable predictions in regions of the input space which have not been covered sufficiently in the training dataset, i.e. the inability to successfully perform extrapolation. The performance of the inverse controller based on RBF networks described by (10), could be gravely affected by extrapolation, even if special care is taken to collect training data that cover sufficiently the input space in terms of the state variables. The reason is that during the network training phase, each input vector does not include a setpoint value, but rather the next value of the controlled variable; the latter is substituted by the setpoint $\omega(k)$ only during the controller operation. Therefore, depending on the relative values of $\mathbf{x}(k)$ and $\omega(k)$ and the sampling time of the controller, it may be infeasible to move the system from its current state value $\mathbf{x}(k)$ to the desired setpoint value $\omega(k)$, within a single discrete time period. In such a case, the neural controller will inevitably be asked to extrapolate, as it is obvious that no similar examples have been used during the training phase of the RBF network.

The inclusion of a concept widely used in chemometrics, known as the applicability domain (AD) of the model, has been shown to improve the performance [9] of inverse NN controllers. AD is used in order to give an indication on whether a model performs extrapolation and thus characterize the reliability of the model prediction. According to the AD criterion, the following equation defines the marginal condition for avoiding extrapolation:

$$\left[\mathbf{x}(k) \quad \omega(k)\right] \cdot \left(\mathbf{U}^T \cdot \mathbf{U}\right)^{-1} \cdot \left[\mathbf{x}(k) \quad \omega(k)\right]^T = 3\frac{N+1}{K} \tag{11}$$

where N is the number of input variables, K is the number of training data and $\mathbf{U}$ is a matrix containing all input training data. Eq. 11 is a second-order equation with respect to the current setpoint value $\omega(k)$. Its two solutions $\omega_{\min}(k)$ and $\omega_{\max}(k)$ actually define the lower and upper bounds on the value of $\omega(k)$ that guarantee that extrapolation is avoided. In many cases it is desirable to tighten those limits, e.g. in order

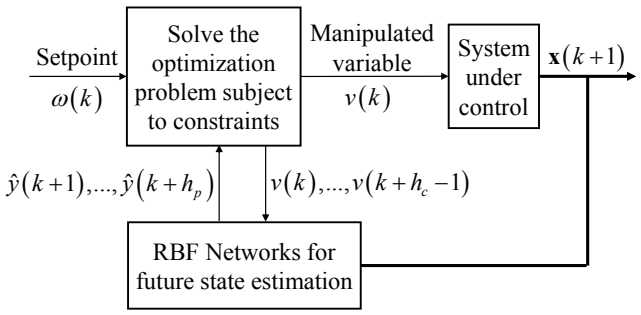

Fig. 2. Integration of RBF model to the MPC controller

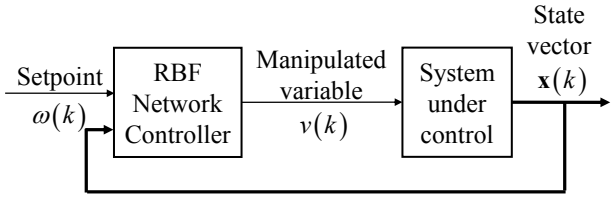

Fig. 3. A direct neural controller based on an inverse RBF model

to account for inaccuracies in the training data. This is accomplished by dividing the length of the line segment between $\omega_{\min}(k)$ and $\omega_{\max}(k)$ by a narrowing parameter r, where r>1. Thus, a narrower AD is produced whose bounds are given by:

$$\omega'_{\min}(k) = \frac{(r+1)\omega_{\min}(k) + (r-1)\omega_{\max}(k)}{2r}$$
$$\omega'_{\max}(k) = \frac{(r+1)\omega_{\max}(k) + (r-1)\omega_{\min}(k)}{2r} \quad (12)$$

In cases where the setpoint $\omega(k)$ falls outside of the narrowed limits calculated by (12), it is substituted by the closest of these limits, so as to keep the input vector inside the narrowed AD. Thus, a requested setpoint value $\omega_{RE}(k)$ replaces the original one and the control law becomes:

$$v(k) = \mathrm{RBF}\left(\mathbf{x}(k), \omega_{\mathrm{RE}}(k)\right) \quad (13)$$

The narrowing parameter is also used as a means for tuning the neural controller, as larger values of r imply that control actions will be more conservative, whereas smaller values of r will trigger more aggressive control actions.

IV. Case Study: Control of a Nonlinear CSTR

A. System description

This section presents a comparison between INCAD and the neural-based MPC (NMPC) controller described in the previous section. Both controllers are evaluated on the control of a simulated non-isothermal CSTR, where an exothermal irreversible reaction between sodium thiosulfate and hydrogen peroxide is taking place:

$$2Na_2S_2O_3 + 4H_2O_2 \rightarrow Na_2S_3O_6 + Na_2SO_4 + 4H_2O \quad (14)$$

This process is described by the following mass and energy balances [13]:

$$\frac{dC_A}{dt} = \frac{F}{V}\left(C_{A,in} - C_A\right) - 2k_o \exp\left(-\frac{E}{RT}\right)C_A^2$$
$$\frac{dT}{dt} = \frac{F}{V}\left(T_{in} - T\right) + 2\frac{(-\Delta H)_R}{\rho c_p}k_o \exp\left(-\frac{E}{RT}\right)C_A^2 - \frac{UA}{V\rho c_p}(T - T_j) \quad (15)$$

where V is the volume of the CSTR; $(\Delta H)_R$ is the heat of the reaction; $-E/R$, k_o, c_p, ρ are constants of the reaction and the reactants; F is the flow rate into the reactor; $C_{A,in}$ is the inlet concentration of the reactant $Na_2S_2O_3$; C_A is the concentration of $Na_2S_2O_3$ inside the reactor; T_{in} is the inlet temperature; T is the temperature inside the reactor; T_j is the temperature of the coolant and UA is the overall heat-transfer coefficient between the cooling coil and the reactor contents. The values of the process parameters can be found in [9].

It is well known that this type of CSTR has normally three steady-state points, an upper and a lower one which are stable, and a middle one which is unstable. Though it is relatively easy to control the CSTR around the individual upper or lower steady-state point, controlling the CSTR around the unstable steady-state point is a rather challenging task.

The CSTR was simulated by solving the system of ODEs in (15). The objective was to apply the proposed neural controller in order to control the concentration C_A inside the CSTR using the temperature of the coolant T_j as the manipulated variable, assuming that both state variables (C_A, T) can be measured in real time.

B. RBF model training

In order to generate data for training the RBF models needed for the two controllers, random changes from a uniform distribution were applied to the coolant temperature T_j every 0.5 seconds, within the limits 0 – 500. Using the described configuration, 60000 data points were collected from the CSTR. The data points were split into a training dataset of 42000 data points and a validation one of 18000 data points.

In the case of the indirect NMPC controller, the RBF model is set up so as to predict the value of the controlled variable at the next time instant, given the full state vector and the manipulated variable at the current time instant k:

$$C_A(k+1) = \mathrm{RBF}\left(C_A(k), T(k), T_j(k)\right) \quad (16)$$

Eq. (16) indicates that in order to calculate future values of C_A throughout the prediction horizon, future values of the second state variable T should be used as inputs, e.g. prediction of $C_A(k+2)$ requires an estimation of $T(k+1)$. Therefore, a second RBF model is also trained using the same input variables:

$$T(k+1) = \mathrm{RBF}\left(C_A(k), T(k), T_j(k)\right) \quad (17)$$

After collecting the training data, an exhaustive search was performed, testing partitions ranging from 4 to 100 fuzzy sets. In each case, the best network was selected based on the Root Mean Square Error (RMSE) value on the validation dataset. Table I shows the results corresponding to the best network found for the two state variables, including the number of fuzzy sets, RBF centers and the RMSE and R^2 pointers on the validation dataset. It should be noted that the statistical pointers used in this paper for facilitating comparison between the two controllers are defined in [9].

As far as the INCAD controller is concerned, an inverse model was trained; the inverse model predicts the value of the manipulated variable $T_j(k)$ that must be applied at the current time instant, given the current state vector, so that the concentration in the next time instant will be equal to

TABLE I
Specifications And Statistics for the Best Trained RBF Network

Parameters	NMPC C_A model	NMPC T model	INCAD inverse model
Fuzzy Partition	34	34	88
RBF Centers	449	449	726
RMSE validation	0.005	0.788	18.76
R^2 validation	0.999	0.999	0.965

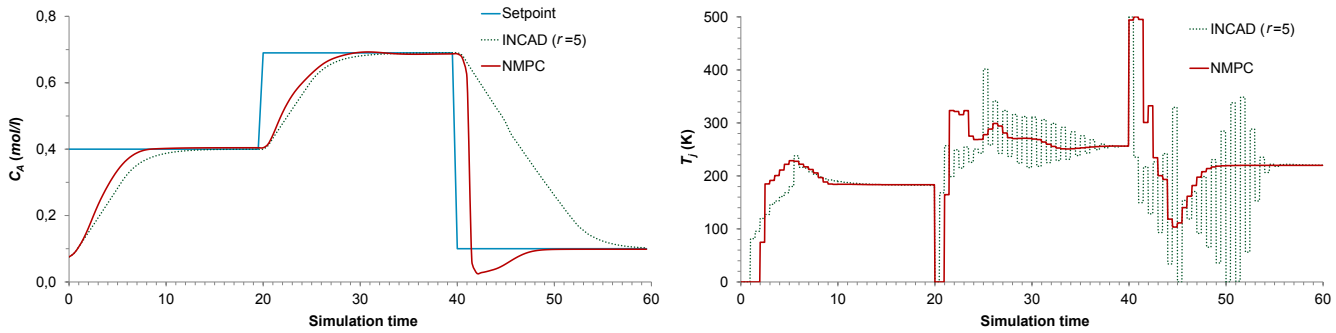

Fig. 4. Setpoint tracking problem: (a) Controller responses, (b) Controller actions

$C_A(k+1)$:

$$T_j(k) = \mathrm{RBF}\left(C_A(k), T(k), C_A(k+1)\right) \quad (18)$$

Although model (18) was trained on the same collected data with models (16-17), it was found that a significantly denser fuzzy partition was needed in order to produce the best RMSE value, resulting to a higher number of RBF centers. The RMSE and R^2 pointers found for the inverse model case, indicate a lower accuracy compared to the accuracy achieved for the two state prediction models used by MPC. This can be attributed to the nature of the inverse model function g, which, in the general case is expected to be more difficult to approximate compared to the function f representing the state prediction model.

C. Controller tuning

Tuning for both controllers was performed using a trial and error procedure. In both cases, the objective was to achieve the fastest possible response without generating large overshoots. Regarding the NMPC controller, the size of the control horizon was set equal to 5 time steps, as longer horizons were found to increase the computational load of the optimization problem, without improving the response. The prediction horizon was set so as to give enough time to the CSTR to reach equilibrium after imposing the last control move. The error and move suppression weights were selected so as to avoid oscillations, without significantly delaying system response.

As far as the INCAD controller is concerned, the tuning procedure boils down to the selection of a single parameter, i.e. the narrowing parameter r. A value of $r = 5$ was selected, as it was found to provide a balance between aggressive behavior with fast but oscillatory response, to conservative behavior with no oscillations, but slower response. The parameters used for both controllers are summarized on Table II.

TABLE II
CONTROLLER PARAMETERS

Parameter	Symbol	Value
NMPC Controller		
Error weights	Ω	1500
Move suppression weights	Θ	1
Control horizon size	h_c	5
Prediction horizon size	h_p	20
INCAD Controller		
Narrowing parameter	r	5
Both Controllers		
Manipulated variable lower bound	v_{min}	0
Manipulated variable upper bound	v_{max}	500

D. Results and Discussion

The performance of the two control schemes was evaluated in a setpoint tracking problem. To be more specific, a test comprising a series of setpoint changes was improvised, aiming to cover the middle unstable steady state point, as well as other parts of the CSTR operating region.

The responses for NMPC and INCAD, together with the setpoint changes are shown in Fig. 4a, whereas Fig. 4b depicts the respective control actions. The mean absolute error (MAE), mean absolute steady-state error (MASSE), mean control action calculation time (MCACT) and maximum control action calculation time (MaxCACT) for both control schemes are summarized on Table III. Computational times were measured on a PC with an Intel® Core™ i7 processor (2.9 GHz) and 8GB of RAM. It can be seen that both controllers succeed in driving the CSTR successfully through the applied setpoint changes, including the region around 0.7, which corresponds to the unstable equilibrium point.

The INCAD controller exploits the inverse model, while the incorporation of the narrowed AD criterion helps to eliminate extrapolation. The result is a smooth response, avoiding any overshoots. On the other hand, the benefits of incorporating an optimization procedure to the NMPC control framework are manifested through faster responses and a significantly lower MAE value compared to the INCAD controller. INCAD just calculates one of possibly many feasible actions that moves the system towards the setpoint, whereas NMPC performs a search among candidate sequences

TABLE III
SIMULATION RESULTS

Parameter	NMPC	INCAD
MAE	0.06	0.11
MASSE	0.0009	0.0002
MCACT (*s*)	0.75	0.0015
MaxCACT (*s*)	3.79	0.0018

of moves to find the optimal course of action. It should be noted, however, that although NMPC is the first to reach the setpoint in all the applied changes, it exhibits a high overshoot at the final setpoint change.

Another benefit from NMPC derived from the formulation of the optimization problem, is related to the control actions. It can be seen that following each setpoint change, the INCAD controller initially chooses oscillatory control moves which progressively decay to a steady value. NMPC on the other hand, incorporates a move suppression term in the objective function, which helps to avoid excessive oscillations in the control moves. It should be noted, however, that NMPC implementation requires to tune a significantly increased number of parameters compared to INCAD.

As far as steady-state error is concerned, both controllers manage to reach near the setpoint values, however, the INCAD controller results to much lower MASSEs than NMPC. This is despite the fact that the inverse model used by INCAD exhibits lower accuracy compared to the NMPC predictive models. This could be attributed to the fact that the models used by NMPC perform multiple-step ahead prediction, which magnifies the prediction error, but also to the incorporation of the AD by INCAD, which helps the latter to reduce modelling error by avoiding extrapolation.

Finally, an important factor, especially related to real-world implementation of neural controllers, is the time needed for calculating the control action. In this field, the INCAD controller has the absolute advantage, as its output is the result of one function evaluation. In fact, INCAD never exceeds 1.8ms in calculating the control action. The average time needed by NMPC to calculate the control actions is orders of magnitude higher (0.75s) due to the computational burden imposed by solving the optimization problem. To make things worse, NMPC calculations become much slower in time instances when a setpoint change occurs, and can reach up to 3.79s. This can be explained taking into account that NMPC optimization uses as an initial guess the solution generated by the solver during the previous time instant. However, when the setpoint changes, possibly a radical alteration in the previously calculated course of action is needed, significantly delaying the solution of the optimization problem.

V. Conclusion

This work presents a comparison between two control schemes based on RBF networks, namely a novel direct inverse neural controller based on the AD criterion and an indirect MPC controller using neural models to acquire predictions for a future horizon. A highly nonlinear system with unstable equilibrium points is used as benchmark. Results show that the NMPC controller exhibits faster responses with less oscillatory control moves. The INCAD controller, however, exhibits lower steady state errors, less overshoot and more importantly is able to calculate the control action in orders of magnitude less time compared to NMPC. Therefore, INCAD can provide satisfactory control performance, while at the same allowing for very fast control action calculation, a fact which can expand its real-world implementation potential to systems with fast dynamics, as opposed to NMPC.

Future research plans include more extensive testing of the two methodologies on different control problems including disturbance rejection and unmodeled dynamics, as well as application on different nonlinear systems.

References

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ: Prentice Hall International, 1999.

[2] P. J. Antsaklis. (1990, Apr.) Special Issue on 'Neural Networks in Control Systems'. *IEEE Contr. Syst. Mag.* 3-87.

[3] K. J. Hunt, D. Sbarbaro, R. Zbikowski and P. J. Gawthrop, "Neural networks for control systems - A survey," ***Automatica,*** vol. 6, pp. 1083-1112, 1992.

[4] A. Alexandridis, H. Sarimveis and G. Bafas, "A new algorithm for online structure and parameter adaptation of RBF networks," ***Neural Networks,*** vol. 16, pp. 1003-1017, Sep. 2003.

[5] M. Lawrynczuk, "Training of neural models for predictive control," ***Neurocomputing,*** vol. 73, pp. 1332-1343, May 2010.

[6] A. Alexandridis and H. Sarimveis, "Nonlinear adaptive model predictive control based on self-correcting neural network models," ***AICHE J,*** vol. 51, pp. 2495-2506, 9 Jun. 2005.

[7] T. Waegeman, F. Wyffels and B. Schrauwen, "Feedback Control by Online Learning an Inverse Model," in ***IEEE T Neural Net Learn Syst***, vol. 23, pp. 1637-1648, 2012.

[8] J. Chia-Feng and C. Jung-Shing, "A Recurrent Fuzzy-Network-Based Inverse Modeling Method for a Temperature System Control," ***IEEE T Syst. Man Cy. C,*** vol. 37, pp. 410-417, May 2007.

[9] A. Alexandridis, M. Stogiannos, A. Kyriou and H. Sarimveis, "An offset-free neural controller based on a non-extrapolating scheme for approximating the inverse process dynamics," ***J Process Contr.,*** vol. 23, pp. 968-979, Aug. 2013.

[10] C. Darken and J. Moody, "Fast adaptive k-means clustering: some empirical results," in *International Joint Conference on Neural Networks (IJCNN)*, San Diego, CA, USA, 1990, pp. 233-238.

[11] A. Alexandridis and E. Chondrodima, "A medical diagnostic tool based on radial basis function classifiers and evolutionary simulated annealing," ***J Biomed. Inform.,*** vol. 49, pp. 61-72, Jun 2014.

[12] A. Alexandridis, "Evolving RBF neural networks for adaptive soft-sensor design," ***Int. J Neural Syst.,*** vol. 23, p. 1350029, 20 Aug. 2013.

[13] N. Kazantzis and C. Kravaris, "Synthesis of State Feedback Regulators for Nonlinear Processes," ***Chem. Eng. Sci.,*** vol. 55, pp. 3437-3449, 2000.