Optimization of Recipe Based Batch Control Systems Using Neural Networks

A. Šoštarec,[a,*] D. Gosak,[c] and N. Hlupić[b]
[a]Pliva Croatia Ltd., Prilaz baruna Filipovića 25, Zagreb
[b]University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Applied Computing, Unska 3, Zagreb
[c]present address: Hospira Zagreb Ltd., Prilaz baruna Filipovića 27, Zagreb

Original scientific paper
Received: May 25, 2012
Accepted: August, 22, 2012

In the modern pharmaceutical industry many flexible batch plants operate under an integrated business and production system, using ISA S95 and ISA S88 standards for models and terminology, and implementing flexible recipe-based production.

In the environment of constantly changing market conditions, adjustment to surroundings is a business necessity. To support necessary production improvement, regulatory authorities have introduced the risk based approach for the control of process development, production based on the quality by design (QbD) principle, and process analytical technology (PAT).

In this work, the method for practical implementation of an adaptable control recipe, that allows process improvement inside the previously established design space, is proposed, based on the neural network process model.

Based on the neural network model, the three methods for recipe-controlled process improvement and optimization were introduced – neural-based software sensor, generic neural model control, and process optimization using iterative dynamic programming.

Suitability of the proposed method was tested in a mini reaction plant Chemreactor Büchi, running the wastewater treatment batch, controlled by the production recipe based on S88 standard.

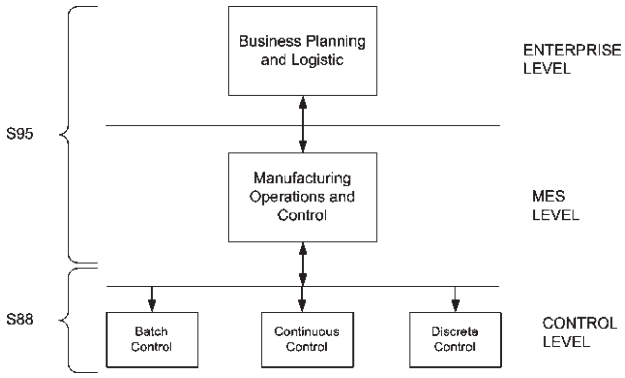
Key words:
ISA S88, neural network model, recipe-controlled process, design space

Introduction

Recipe-based automated flexible pharmaceutical plants

Modern pharmaceutical production is a branch of industry faced with many different challenges. It is characterized by the necessity to maintain high productivity and flexibility, while at the same time complying with strict regulations imposed by different government agencies (FDA, EMEA, etc.). Therefore, to maintain their competitive edge, many pharmaceutical companies (API, as well as final forms producers) use various organizational and process control software tools. Many of the available support and process control tools are based on ISA S95 and S88 standards.[1,2]

Flexible pharmaceutical plants, organized in accordance with integrated ISA S95 – S88 strategy, have the general structure depicted in Fig. 1. While the various S 95 levels are used for a structured approach to business planning and logistics, as well as manufacturing operations and control (so-called MES level – manufacturing execution system level), the S88 standard is used on the lowest level



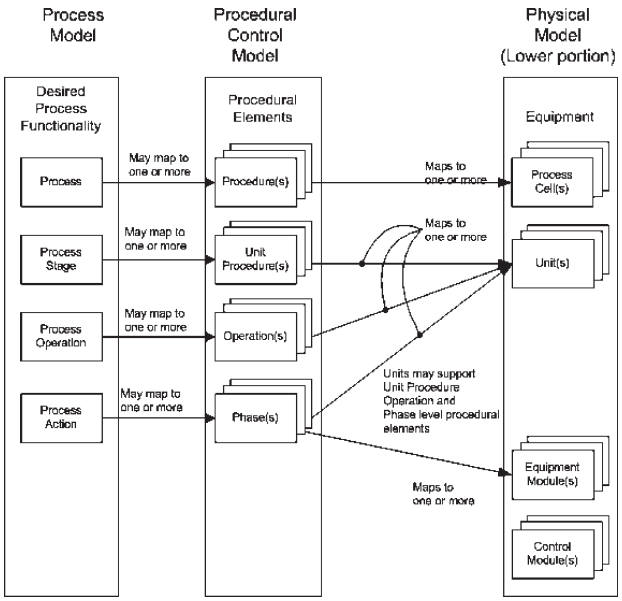
F i g . 1 – *ISA S95 and S88 integration through enterprise, MES and control levels as defined in corresponding standards*

to actually control the production process within a particular plant.

ISA S88.01 models and terminology (1995 and 2011) are created to fulfill the task of unifying terminology and models, so that the experts from different fields (automation, equipment and technology) can easily cooperate. The standard, as a basis, defines the recipe which uniquely specifies the ingredients and steps necessary to produce a certain product. Applying S88 requires separation of physical equipment

*Corresponding author: anita.sostarec@pliva.hr

from production procedures. This is achieved by applying three models: the process model, physical model and procedural control model. The models and their mapping are depicted in Fig. 2. The purpose of mapping is to tie the procedural control model to the physical model, in order to provide processing described in the process model.



F i g . 2 – *Mapping of models according to ISA S88.01 standard*

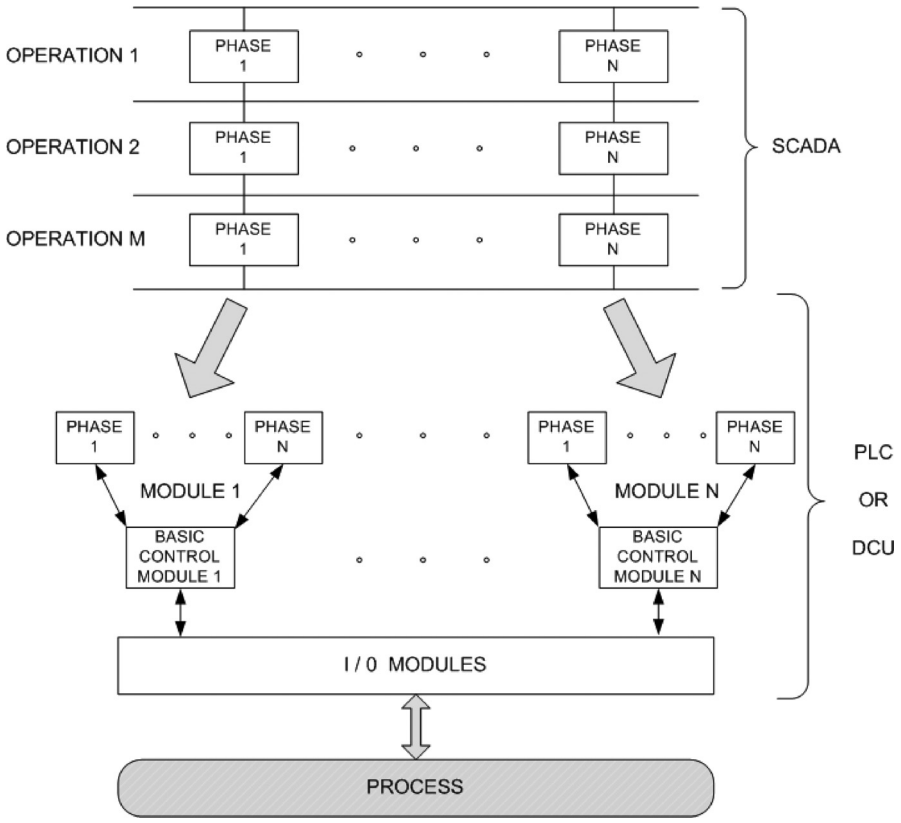
Practical implementation of S88 models, after performing of mapping on existing process equipment, is usually done as depicted in Fig. 3.

Quality by design and PAT methodology

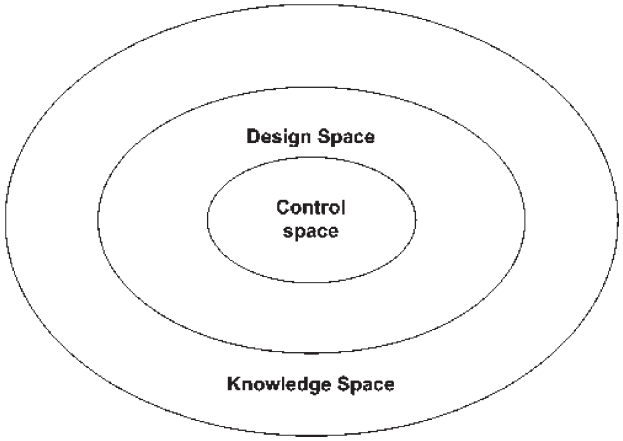
Application of the mentioned methods provides clear benefits, such as improved process reproducibility, lower production costs and increased plant flexibility for introduction of new products. In spite of all this, until recently, process optimization of a particular product was an expensive and time-consuming task due to regulatory requirements that effectively hinder process changes necessary for improvement.

In order to encourage process improvement and, at the same time, ensure high regulatory (GMP – good manufacturing practice) levels in the flexible pharmaceutical production, the FDA has introduced two initiatives – quality by design (QbD)[3] and process analytical technology.[4]

The key concept in QbD is the design space – the established range of process parameters inside of which product quality does not diminish; therefore, movement within these boundaries for optimization purposes is a regulatory allowed change. As can be seen from Fig. 4, the control space – the working process space that spans around process set points – can generally be moved within the de-



F i g . 3 – *Common way of practical realization of ISA S88 models on control equipment*



F i g . 4 – *Positioning of parameter space according to quality by design initiative*

sign space using some optimization technique (numerical or experimental design method) in search of the optimum point.

Another important, recently-introduced paradigm is the process analytical technology (PAT) initiative. The essence of PAT initiative is to allow replacement of classical sampling by off-line analysis with timely (on-line) process parameters measurement, via analyzers as well as applying software sensors when appropriate.

This paper proposes a method based on the neural network model that extends the ISA S88 methodology-based control recipe mapped on a particular physical equipment unit, in a way that allows gradual movement of control space toward the optimal point, as well as application of software sensors.

Flexible batch plants as hybrid systems

As can be seen from available literature (Sanchez *et al.,*[5] Moor and Raisch,[6] Potočnik *et al.*[7]), flexible batch plants represent so-called hybrid systems. Hybrid systems are broad classes of systems whose behavior is characterized by exhibiting continuous and discrete process dynamics. While there are many mathematically rigorous descriptions of such systems (Goebel *et al.,*[8] Lennartson *et al.*[9]), the state space approach of Barton, Banga and Galan[10] is here presented. According to their definition, a general hybrid system can be described by state space $S = \bigcup_{k=1}^{nk} S_k$ where every mode S_k is characterized by:

– Set of variables $\left\{\dot{\boldsymbol{x}}^{(k)}, \boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{u}^{(k)}, t\right\}$

– Set of equations

$$\boldsymbol{f}^{(k)}(\dot{\boldsymbol{x}}^{(k)}, \boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{u}^{(k)}, t) = 0$$

– Set of transitions possibly containing transition conditions $\boldsymbol{L}_j^{(k)}(\dot{\boldsymbol{x}}^{(k)}, \boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{u}^{(k)}, t) = 0$ formed by logical propositions that trigger switching when they become true, and transition functions $\boldsymbol{T}_j^{(k)}(\dot{\boldsymbol{x}}^{(k)}, \boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{u}^{(k)}, \dot{\boldsymbol{x}}^{(j)}, \boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)}, \boldsymbol{u}^{(j)}, t) = 0$ associated with particular conditions

– Set of constraints that could be path constraints $\boldsymbol{h}^{(k)}(\dot{\boldsymbol{x}}^{(k)}, \boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{u}^{(k)}, t) \leq 0$, and point constraints $\boldsymbol{c}_r^{(k)}(\dot{\boldsymbol{x}}^{(k)}, \boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}, \boldsymbol{u}^{(k)}, t_r) \leq 0,\ r \in \{0, \ldots, n_r^{(k)}\}$ that must be satisfied only in certain moments.

Based on hybrid system methodology, Lennartson *et al.*[9] propose the two approaches to practical realization of the hybrid process control. The first approach is to replace hybrid plant model with fully discrete model. This task is performed by partitioning the continuous state space into regions, and then creating the supervisor for apparent discrete process. The main advantage of this approach is easy implementation, while the main disadvantage is possible poor approximation of continuous process parts. The second approach is the direct synthesis of hybrid supervisor controller. The main advantage of this approach is the better control that can be achieved, but at the expense of applying much more complex model.

Another method for realization of hybrid process control is proposed by Sanchez *et al.*[5] Their method is applicable to hybrid systems with predominant discrete parts. The basis of this method is the division of an overall plant model into the process model and discrete event controller model. The process model is further divided into equipment models and interface models. Each equipment model describes the hybrid dynamic of one process unit. The second part of the process model links the unit operation model with the discrete event controller models. It is also made of two separate models: device drivers and state observers. Their role is to connect equipment with the discrete controller and to model discrete events instruments. Supervising the discrete event controller includes an interface for connection with lower levels, and actual implemented control system. Controller is designed to be in compliance with the ISA S88 standard, and therefore consists of phases, operations, and unit procedures.

Recently, Fabre *et al.*[11] used PrODHyS hybrid simulation environment coupled with scheduling module ProSched in order to model batch processes (ISA S88 compliant), and optimize interaction between scheduling and simulation models. Recipes are modeled with extended resource task networks, while scheduling is performed by mixed integer linear programming.

Process modeling using neural networks

Developing the physical, thermodynamically-based mathematical model of a complex pro-

cess is a difficult and time-consuming activity. In case of the flexible batch plant that produces many different products, delays between switching products must be minimized for economic reasons, while at the same time optimization is needed. Therefore, faster ways of developing process models are desired. If only experimental data are available, using the neural network model is a practical option.

Chemical engineering unit operation processes, such as reaction, distillation, crystallization etc. can be generally described with a nonlinear dynamical relation that relates state coordinates $\boldsymbol{x}$ with process inputs $\boldsymbol{u}$ and outputs $\boldsymbol{y}$ (eqs. 1 and 2)

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{1}$$

$$\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \tag{2}$$

Static input – output relation (eq. 2) can be directly approximated by neural network if data for learning are available. In the literature,[12] the neural network which is most often used, is a multilayer feedforward perceptron because of its universal learning capability. Multilayer feedforward perceptron neural network is depicted in Fig. 5, and eqs. 3–5 provide its mathematical representation.

$$x_p = f_p(net_p) \tag{3}$$

$$net_p = \sum x_{p-1} w_{p-1\,p} + w_{0\,p-1} \tag{4}$$

$$f_p(net_p) = \frac{1}{1 + e^{-net_p}} \tag{5}$$

Learning the net means adjusting the weight coefficients in a way that approximation error (eqs. 6 and 7) is minimized.

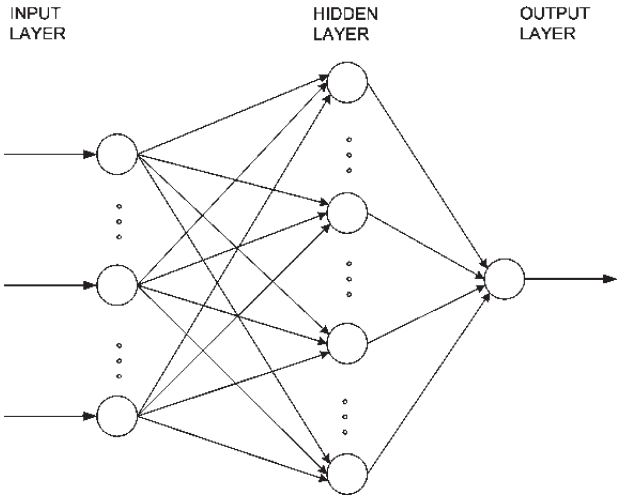


Fig. 5 – *Schematic representation of multilayer feedforward network*

$$E_t = \frac{1}{2} \sum_{i=0}^{I} (y_{ri} - O_{ri})^2 = \frac{1}{2} \sum_{i=0}^{I} e_i^2 \tag{6}$$

$$E_T = \sum_{r=1}^{R} E_t \tag{7}$$

Minimization of the weight coefficients must be done by some minimization method. The most basic is the method of steepest descent (eq. 8)

$$w_{pp-1}(i+1) = w_{pp-1}(i) - \mu_1 \frac{\partial E_T}{\partial w_{pp-1}} \tag{8}$$

Many more sophisticated methods can be applied; deterministic like Levenberg Marquard or conjugate gradient (Masters[13]) as well as stochastic, for example genetic algorithm. The necessary partial derivatives of errors with regard to weight coefficients can be calculated analytically by using, so-called backpropagation algorithm. eqs. 9–12

$$\frac{\partial E_t}{\partial w_{ij}} = -e_i \frac{\partial x_i}{\partial (net_i)} \frac{\partial (net_i)}{\partial w_{ij}} \tag{9}$$

$$\frac{\partial E_t}{\partial w_{jk}} = -\frac{\partial (net_j)}{\partial w_{kj}} \frac{\partial x_j}{\partial (net_j)} \sum_{i=0}^{I} e_i \frac{\partial x_i}{\partial (net_i)} \frac{\partial (net_i)}{\partial x_j} \tag{10}$$

$$\frac{\partial (net_p)}{\partial x_{p-1}} = w_{pp-1} \tag{11}$$

$$\frac{\partial x_p}{\partial w_{pp-1}} = x_{p-1} \tag{12}$$

Many modifications of the presented basic algorithm are developed in order to improve convergence and speed up the computation. While training the net, one must bear in mind that if the network is chosen to have more neurons that are actually necessary to learn a particular relation, it can often overfit the data. The easiest method to prevent this from happening is to divide the entire data set into a training set and a verification set. During the net learning minimization results of training set are checked against the validation set and minimization routine is stopped at the optimal point.

Regarding the learning of the dynamical relation (eq. 1) two main approaches are developed. The first one is the so-called tapped delay line method (Hrycej,[14] Paengjuntuek *et al.*[15]). The principle of this method is to use a static network, but with additional inputs of past values for network inputs as well as outputs (eq. 13)

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}^{NN}(\boldsymbol{x}_t, \ldots, \boldsymbol{x}_{t-d}, \boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t-d}) \tag{13}$$

In eq. 13 d is the measurement index, a number which determines how many past network inputs and outputs must be used in order to approximate eq. 1 with sufficient accuracy. The clear advantage of this method is the ability to apply the static network, but quality of approximation is strongly dependent on the time step and choice of d value.

Another general way for achieving the dynamic process learning by neural network is to use recurrent networks. The recurrent networks may be internally or externally recurrent. Internally recurrent nets are nets whose neurons are fully connected with each other, as well as with themselves. Activation functions of these kinds of neurons are differential equations (eq. 14)

$$\tau_p \frac{\mathrm{d}x_p}{\mathrm{d}t} = -x_p + f_p(net_p) \qquad (14)$$

To train this kind of network, two principal groups of methods can be applied (De Jesus and Hagan[16]). The first group is called Backpropagation Trough Time (BPTT) and its main principle is developed by Werbos[17] who introduced the idea to discretize eq. 14 around current point t_i, using a method for calculation of finite differences. After adjusting backpropagation equations to reflect the new expression for neuronal activation function, calculation is performed backwards in time for a number of past inputs appearing in discretized eq. 14. The second group of methods is called Real Time Recurrent Learning (RTRL) originally proposed by Williams and Zipser.[18] The main characteristic of this method is learning in real time, i.e. while network is running by integration of neuron activation function (eq. 14). The network weights are updated after every integration step.

The internally recurrent neural network learning process suffers from a problem called vanishing gradient (Haykin[12]). This problem makes learning of the behavior that depends on the data belonging to a distant past, with respect to the current time, very difficult due to the local nature of both learning methods. Therefore, another kind of recurrent network, namely externally recurrent neural networks, are introduced. The simplest form of externally recurrent neural network is introduced by Nerrand *et al.*[19] (eq. 15) and called the canonic form of recurrent network.

$$\boldsymbol{x}(i+1) = \boldsymbol{x}(i) + \boldsymbol{f}^{NN}[\boldsymbol{x}(\boldsymbol{i}), \boldsymbol{u}(\boldsymbol{i})] \qquad (15)$$

While the network given by eq. 15 is taught by the standard algorithm, and thus does not suffer from the vanishing gradient problem, the practical performance of the net can be less than satisfactory due to the fact that applied simple discretization may not be sufficient in the case of approximation of highly nonlinear systems, especially if they belong to the class of "stiff" dynamical systems. For this reason, Gosak and Vampola[20] used continuous canonic form (eq. 16).

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}^{NN}(\boldsymbol{x}, \boldsymbol{u}) \qquad (16)$$

In order to perform the task of learning the recurrent network described by eq. 16, derivatives of measured state vector data must be calculated at each point, and the quality of approximation heavily depends on the chosen numerical method. Additionally, resulting continuous differential equation model must be solved by integration. Despite all this, the resulting network may be the network of choice because of its ability to cope with difficult nonlinear system, by enabling any method of integration – multistep, semi implicit or implicit to be applied.

Methodology

Outline of neural network-based extension to recipe-based control

A recipe-based flexible plant is normally designed to produce many different products with only two common characteristics – allowable ranges of process parameters (temperature, pressure etc.) and equipment construction materials. Usual design procedure includes several steps, such as defining the process units equipment modules, defining the basic control modules, and set of equipment phases designed to perform predefined module transitions. Whenever a new product is introduced to the production plant, only product-specific control recipe needs to be developed.

While this design procedure allows quick product switching and high batch reproducibility, the fact that the resulting control system is not in any way built in for any particular product introduces potential drawbacks. The main drawbacks are the following: lack of specific sensors needed for on-line measurement (e.g., concentration, density, conductivity), control loops may suffer from strong interactions, and particular operations may operate far from optimum points.

It is well known that all the mentioned drawbacks could be eliminated by the use of an appropriate process model for the model based optimization and process control improvement. A suitable form of a flexible batch plant model is a hybrid model generally described in the text above, i.e. a model that takes into account the fact that the un-

derlying dynamical process consists of mixed discrete and continuous parts. The main problem, when formal models such as neural networks are used, that depends solely on experimental data is how to perform the successful training. Particularly, it would be very difficult for neural model to learn the exact switching times of a hybrid system process. Therefore, the control recipe should be constructed in a way that a resulting process learning by neural network can be accomplished. One way to do this is to ensure that the batch system behaves as a hybrid automaton. The concept of hybrid automata was introduced by Alur[21] and are defined as the particular form of hybrid systems modeled as a set of continuous dynamic nodes that become active in a predefined order based on some set of discrete events.

In order to unambiguously define the structure of a hybrid automaton model that corresponds to a particular process, it is appropriate to impose some additional constraints on the control recipe.

The first constraint is that the equipment phase behaves in a way that it performs some transition on the equipment module. This transition can generally be described in the form of piecewise linear function for a set point of variables controlled in particular phases (eqs. 17–19)

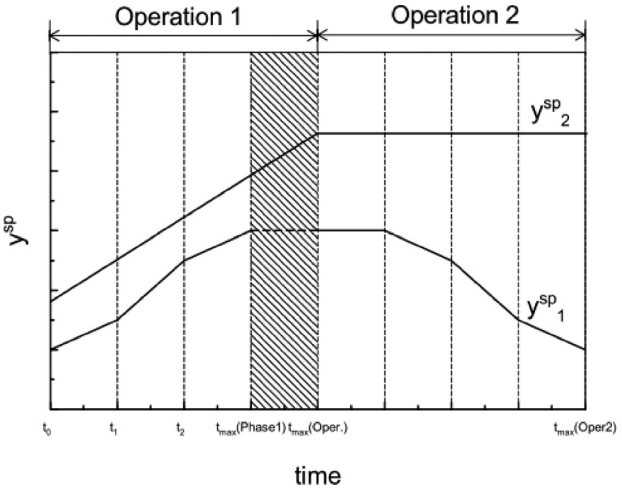
$$l = t - t_k \tag{17}$$

$$\partial_k = \frac{y_{k+1}^{sp} - y_k^{sp}}{t_{k+1} - t_k} \tag{18}$$

$$y^{sp}(t) = \begin{cases} y_k^{sp} + l\partial_k, k = 1, \ldots, K & t < t_{max}^{Phase} \\ y_K^{sp} & t_{max}^{Phase} \le t < t_{max}^{Operation} \end{cases} \tag{19}$$

By applying eqs. 17–19, any transition function with arbitrary precision can be defined, although fixed set points or ramps are mostly used, for example temperature ramp on temperature control module or pressure ramp on pressure control module. After the final value of set point value has been achieved (e.g., final temperature or pressure), the phase remains in this state (with final set point value) until the new operation starts.

The second constraint is that recipe operation represents the set of phases that all start at the same time and operation ends when the last phase in the operation completes its transition, thus ensuring proper ending of continuous nodes. Fig. 6 shows this graphically, with dashed part of a polyline representing a set point of a finished phase waiting for the end of operation.



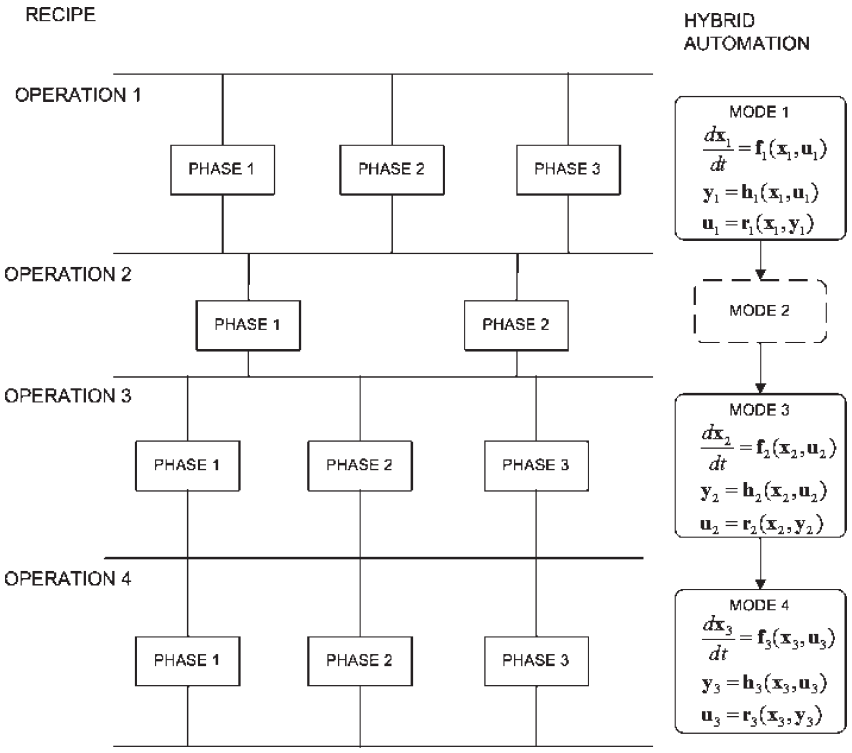
F i g . 6 – *Set point transitions for particular phases in operations. Shaded area shows regions in which phase 1 waits for operation completion by holding the last active set point (dashed line).*

While these additional constraints make development of the hybrid automaton model easier, they do not impose significant limits on recipe capability because, if necessary, any operation can be split into several shorter ones, thereby avoiding creation of time lags.

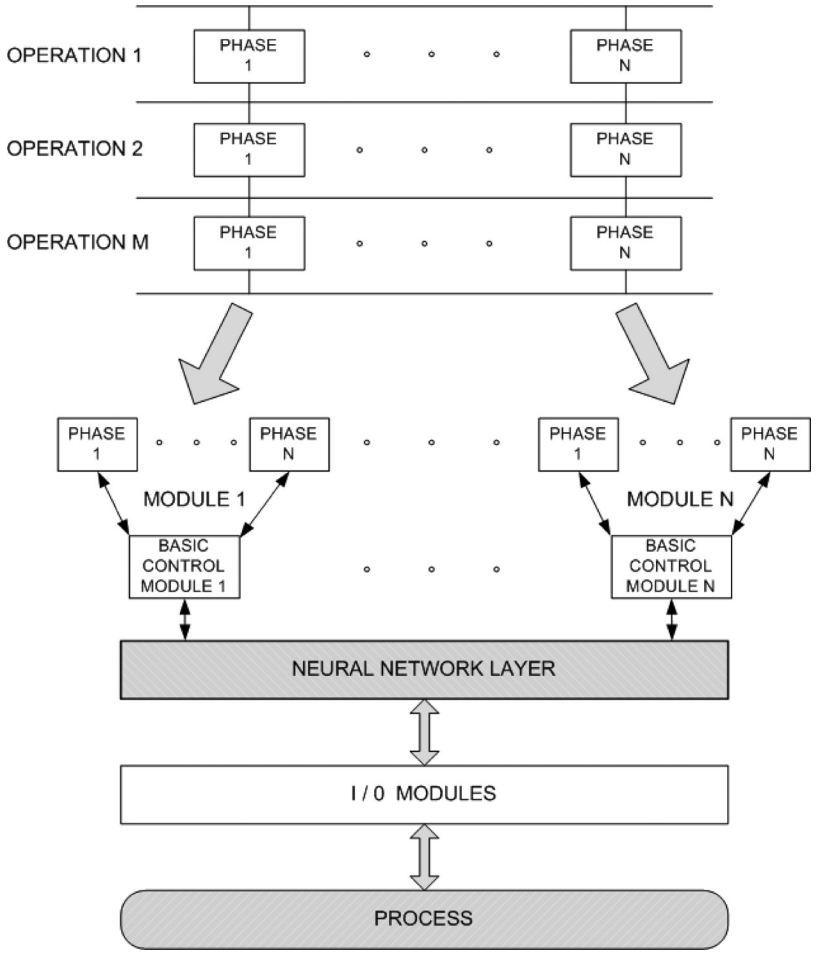
Having in mind all the operations in a recipe – actually interesting for modeling are only those specific to a particular process (reaction, distillation, crystallization), while others are common to all products (inertization, sterilization, cleaning, filling, emptying). Fig. 7 shows an example schematic of a control recipe in usual SFC (sequential function chart) representation, and corresponding hybrid automaton model in a state diagram form (dashed boxes represent common operations not necessary for the model).

After training the neural network for every continuous mode, the network can be used as a software sensor to enhance process control performance, as well as for process optimization. Since the introduction of neural models should be optional, and should only support the existing basic control instead of replacement it, the most suitable way to extend a control system with a neural network is to form a neural process abstraction layer as depicted in Fig. 8.

Detailed structure of a network layer is given in Fig. 9. As can be seen from the figure, the network switches on when modeled operation starts using the appropriate parameter set, and it switches off after the operation with modeled continuous mode finishes. Depending on the task required for the operation neural model, software sensing, controller improvement or optimization of set point curve defined by eq. 19 is performed.



F i g . 7 – *Representation of recipe operations and equivalent modes of hybrid automation (with MODE 2 chosen not to be modeled)*



F i g . 8 – *Modification of recipe control system by insertation of neural layer*

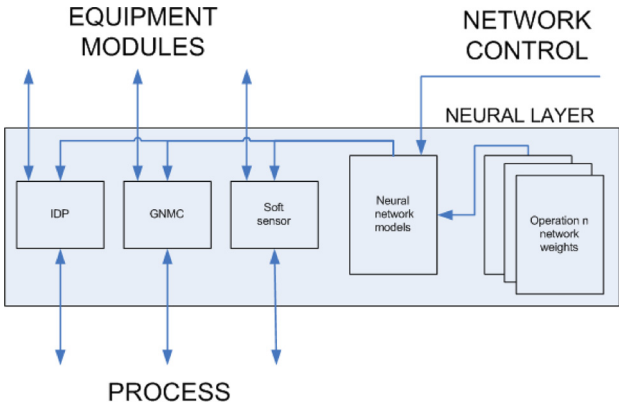


Fig. 9 – *Detailed internal view of neural layer construction*

The development of neural network process model

The neural network needs to model processes on all continuous modes that correspond to operations whose behavior is characteristic for a particular production process. Continuous modes can be generally modeled with equations describing state variable dynamic (eq. 16), output variables relation (eq. 20), and an equation defining controller actions (eq. 21) of involved equipment module basic control.

$$\boldsymbol{y} = \boldsymbol{h}^{NN}(\boldsymbol{x}, \boldsymbol{u}) \tag{20}$$

$$\boldsymbol{u} = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}) \tag{21}$$

Neural networks are needed to approximate eqs. 16 and 20. Both relations are generally nonlinear, and eq. 16 represents the system of differential equations. In order to train the networks, three questions need to be answered: what will be the network topology, how will training data sets be collected, and which training method will be used.

In this paper, for mapping the static relations, the multilayer feedforward neural network (Fig. 5) is chosen. Dynamic process data was mapped by continuous externally-recurrent modification of the multilayered feedforward network.

The next issue is gathering and preparation of a training data set. Since in this particular problem the design space represents the allowable span of process variables, there are two options for data collection – the first is moving the control space inside the design space by changing set points of desired process outputs, and the second way is to extend the control space to the entire design space. In the second case, process inputs may be varied without controller feedback in order to cover the entire design space. Running several trial batches experimentally (without the need to produce a product within the specification) may significantly speed up the data collection process. Of course, data for state variables that are not directly measurable must be collected off-line by taking and analyzing samples during batch runs.

After data collection is performed, the static output relation (eq. 20) can be learned by the neural network. Unfortunately, state space dynamic relations (eq. 16) cannot be learned directly from the collected data, so some form of data preparation must be done. In physics, a well-known method called vector field reconstruction[22] is used for parameter estimation in ill-defined systems. The main principle of the method is to sample data as a time series and then use a numerical method to approximate time derivatives for particular trajectories. The same can be done with collected state space variables data. For this purpose, a smoothing cubic spline polynomial[23] is used (eq. 22) that interpolates the cubic polynomial between each pair of experimental points with additional smoothing criteria given by eq. 23.

$$S_i(t) = a_i(t-t_i)^3 + b_i(t-t_i)^2 + c_i(t-t_i) + d_i \qquad i = 1,\ldots,n \tag{22}$$

$$SSQ = p\sum_i W_i[y_i - S_i(t)]^2 + (1-p)\int \left(\frac{\mathrm{d}^2 S}{\mathrm{d}t^2}\right)^2 \mathrm{d}t \tag{23}$$

After analytical derivation of the cubic polynomials, derivatives on the left-hand side of eq. 16 are obtained for every training point, so static neural network (eqs. 3 – 5) can be used.

The third important decision is the choice of a training method. As already mentioned, the multilayer perceptron network is trained by the use of analytical determination of error gradients – backpropagation algorithm along with parameter to find a numerical method such as the steepest descent, conjugate gradient or Levenberg-Marquard methods. Due to the fact that classical backpropagation algorithm is slowly convergent, in this paper, modification of the basic algorithm, the so-called OWO-HWO method by Manry *et al.*,[24] was applied. This modification allows much faster convergence, but the network must have additional constraints because liner output neurons must be used while the net may have only one hidden layer of neurons.

After the networks have been trained, i.e. their optimal coefficients have been determined, soft sensor for immeasurable coordinates of state vector is obtained for every particular mode (operation) by integrating the dynamical system (eqs. 16 and 20). In practice, mismatch problems may arise due to the mismatch between the actual process and the

network model and/or existence of measurement noise. Therefore, state observer[25] could be used (eq. 24)

$$\frac{\mathrm{d}\hat{\boldsymbol{x}}}{\mathrm{d}t} = \boldsymbol{f}(\hat{\boldsymbol{x}}, \boldsymbol{u}) + \boldsymbol{K}[\boldsymbol{y} - \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{u})] \quad (24)$$

The matrix $\boldsymbol{K}$ represents observer gain. Usually in the case of model mismatch only, the gain is calculated using the extended Luenberger observer algorithm, whereas if process noise is also present, the extended Kalman estimator method for gain calculation is used. In both cases, model gradients need to be calculated and this could be accomplished analytically using a backpropagation algorithm, so this is an additional indication in favor of using a multilayer feedforward network.

Generic neural model control

During the flexible plant design phase, the basic control for particular equipment modules is usually defined as classical PID control, either in a single loop or in cascade. The main problem that could occur in process implementation is control performance deterioration, caused by strong loop interaction (e.g. dosing and temperature). This effect can be minimized by using a neural network model as a generic process model and implementing generic neural model control (GNMC). In the literature, the same method has already been applied by Abonyi *et al.,*[26] and Gosak and Vampola.[20] The method uses a neural model as the particular model in Lee and Sullivan,[27] the well-known generic model control method. The basis of the method is the idea that the output set point dynamic can be maintained at a desired level as shown by expression 25.

$$\left(\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t}\right)^{SP} = \boldsymbol{K}_1(\boldsymbol{y}^{SP} - \boldsymbol{y}) + \boldsymbol{K}_2\int_0^t (\boldsymbol{y}^{SP} - \boldsymbol{y})\mathrm{d}t \quad (25)$$

By the transformation shown in eq. 26, the output can be related to model the following equation

$$\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t} = \frac{\mathrm{d}\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u})}{\mathrm{d}t} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{u}) \quad (26)$$

Finally, the model is introduced in the controller expression as shown in eq. 27

$$\begin{aligned} &\frac{\partial \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u})}{\partial t} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \\ &= \boldsymbol{K}_1(\boldsymbol{y}^{SP} - \boldsymbol{y}) + \boldsymbol{K}_2\int_0^t (\boldsymbol{y}^{SP} - \boldsymbol{y})\,\mathrm{d}t \end{aligned} \quad (27)$$

To apply generic model control, eq. 27 should be solved by control input vector $\boldsymbol{u}$. Depending on model complexity, this can be done analytically or numerically. In the case of a neural network model, the first multiplicand expression on the left-hand side of eq. 27 can be calculated analytically by backpropagation algorithm, and then the resulting equation must be solved numerically. However, there is a special case of GNMC when state variables are directly controlled, either if they are measurable or calculated by software sensor. In this case, eq. 28 is used instead of eq. 25:

$$\left(\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}\right)^{SP} = \boldsymbol{K}_1(\boldsymbol{x}^{SP} - \boldsymbol{x}) + \boldsymbol{K}_2\int_0^t (\boldsymbol{x}^{SP} - \boldsymbol{x})\,\mathrm{d}t \quad (28)$$

Then, eq. 27 equivalent expression is given by eq. 29

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{K}_1(\boldsymbol{x}^{SP} - \boldsymbol{x}) + \boldsymbol{K}_2\int_0^t (\boldsymbol{x}^{SP} - \boldsymbol{x})\,\mathrm{d}t \quad (29)$$

If eq. 28 is the equation of the controller, then in the case of a neural model, instead of the numerical solution of eq. 28, the partial inversion of the neural network can be done by generating a set of data using eq. 16 and then using this data set for training the network (eq. 30) by exchanging places of vectors $\boldsymbol{u}$ and $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ (assuming they have the same dimension).

$$\boldsymbol{u} = \boldsymbol{f}^{NNinv}\left(\boldsymbol{x}, \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}\right) \quad (30)$$

The generic neural model control technique has, in the context of used neural network abstraction layer, a distinct advantage that in eqs. 27 and 29 neural model plays the part of PI controller bias term, therefore no change in equipment modules basic control is necessary; neural model process control enhancement can be switched on and off without changes to the initially designed equipment module or recipe.

Process optimization

The third possibility to enhance the initial recipe with neural model is to use it as a basis for additional optimization. The process inputs that are not engaged in the feedback loop can be used to optimize process performance if a suitable optimization criterion can be set. According to the optimal control theory, a process can be optimized by minimizing a suitable criterion (eq. 31), taking the process model into account, as well as the control input boundaries.

$$\begin{aligned} J[\boldsymbol{x}(t_0)] &= \Psi[\boldsymbol{x}(t_0)] + \int_{t_0}^{t_j} \Phi(\boldsymbol{x}, \boldsymbol{u})\,\mathrm{d}t \\ \beta_i &\le u_i \le \alpha_i \end{aligned} \quad (31)$$

In literature, there are many methods for solving the optimal control problem, but in this work, the iterative dynamic programming method by Luus[28,29] is chosen, because it uses preset boundaries in the search for a solution, and from the computational point of view, it only requires integration of a process model. These characteristics make this method suitable for the present purpose.

The basis of the iterative dynamic programming (IDP) method is to search for P piecewise constant or linear control policy over P time stages of equal time length L. The performance index is thus approximated by eq. 32;

$$J[\boldsymbol{x}(t_0), P] = \Psi[\boldsymbol{x}(t_j)] + \sum_{K=1}^{P} \int_{t_{K-1}}^{t_K} \Phi[\boldsymbol{x}(t), \boldsymbol{u}(t_{K-1})] \mathrm{d}t \quad (32)$$

The algorithm works as follows (Bojkov and Loos):

1. Divide the total time interval into P stages of length L.

2. Choose the number of $\boldsymbol{x}$ grid points N and the number of M allowable values for each control variable.

3. Choose the initial size of a search region $\boldsymbol{r}_i$ for each control variable.

4. Integrate the process model from t_0 to t_f with N evenly distributed control values in the allowable region, thus generating N values of $\boldsymbol{x}$ in each stage.

5. Starting with stage P, for each x grid point integrate the model from $t_f - L$ to t_f with each allowable control value. For each grid point choose control that minimizes the value of J.

6. Move to stage P-1 and repeat the integration from this segment to final time choosing (from step 5) control that gives the nearest $\boldsymbol{x}$ value.

7. Repeat step 6 for all stages.

8. Reduce the size of the search region by γ (eq. 33)

$$\boldsymbol{r}^{(j+1)} = \gamma \boldsymbol{r}^{(j)} \quad (33)$$

9. Increment iteration index j by 1 and go to step 4. Stop the procedure when there is no improvement in minimization of J.

As previously mentioned, the method allows easy checking of search boundaries ensuring that constraints given by design space are satisfied. Also, it uses neural model only for integration (and not for calculation of higher order derivatives and other transformations), which enables the neural network to be used only for the purpose it was trained for. All this increases modeling accuracy.

Experimental

Automated mini plant Chemreactor Büchi

The experimental study was performed in an automated reaction and distillation mini plant Chemreactor C60 produced by Büchi Glass Uster. The control system used for actual method implementation was Siemens PCS7. System PCS7 consists of one AS station (Simatic S7 400 processor) and one OS station, based on Pentium PC computer running MS Windows OS.

Basic control layer was built using Siemens CFC tool. Adjacent phase logic level is programmed using PCS7 specific SCL language. User interface and trend visualization were developed on Siemens WinCC SCADA system. Recipes were implemented[30] using Batch Flexible tool which operates as an add-on to WinCC software. Fig. 10 shows the schematic description of the process unit.

During software design the following equipment modules were implemented:

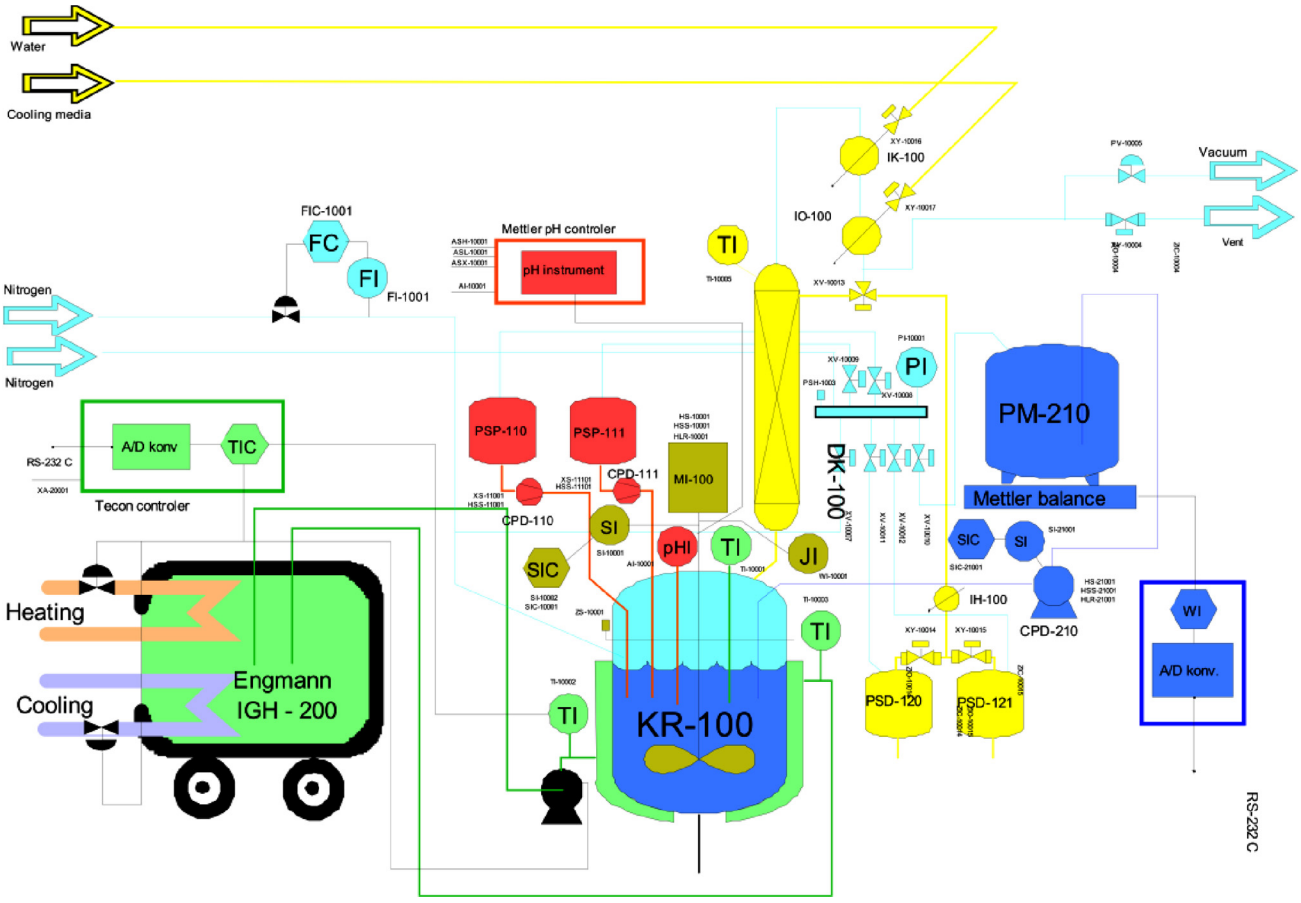
– Heating/cooling module
– Mixing module
– Pressure module
– Distillation module
– pH module
– Dosing module.

For each module a set of equipment phases were created. As an example, Fig. 11 shows phases of Heating/cooling module and the parameters that need to be set for one phase of the module.

By using the method of recipe creation as described in Methodology, new recipes can be created efficiently in a very short time by defining operations, choosing phases for parallel execution in them, and parameterization of phases using SFC editor of Batch Flexible software.

Wastewater treatment by distillation and oxidation

As an example of a batch process that can be constructed using the described equipment, wastewater treatment is performed. This process consists of several operations, i.e., charging of wastewater, purging the equipment with nitrogen, removal of solvent by batch distillation, adjustment of pH value, addition of hydrogen peroxide for detoxification, degradation of the peroxide surplus amount and cooling down the reaction mixture. Fig. 12 shows and example of a complete batch run performed in the test plant with marked operation switches. The purpose of this run was validation of recipe without the use of neural optimization.



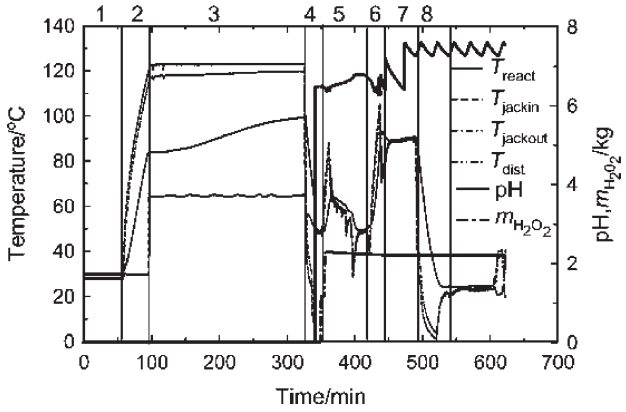
F i g . 1 0 – *Schematic representation of Chemreactor Buchi Uster with defined process modules*

HEATING / COOLING MODULE:

PHASES:
- Initial card
- Reactor temperature
- Jacket temperature
- Reactor / jacket difference
- Jacket in / out difference

PHASE REACTOR-TEMPERATURE PARAMETERS:
- Final temperature
- Temperature gradient
- Max. jacket temperature
- Max. difference

F i g . 1 1 – *Phases of Heating/cooling module implemented on Chemreactor unit and parameters of reactor temperature phase*



F i g . 1 2 – *Wastewater treatment process data log presentation for typical batch with labeled operations*

Testing the process improvement achieved by neural network

While completion of all phases in the recipe is necessary for successful wastewater treatment batch run, improvements can be expected in only two of them – removal of solvent and detoxification by hydrogen peroxide. Other operations are not specific to this process, and are normally optimized during plant startup period.

Normally, the developed method should use data gathered from either on-line or off-line measurement obtained during the batch run. In order to do better testing of the developed neural modeling and optimization method for those two operations or modes of hybrid automation, thermodynamically-based mathematical models were developed using process and literature data, and performing the reaction kinetic determination experiments in the laboratory. Data from the simulation using developed models were then used for training networks.

The solvent removal phase is actually batch distillation of methanol – water mixture intended for the removal of methanol from the wastewater. If the principle of theoretical equilibrium stage is

used, the resulting model is given by eqs. 34–37, while vapor liquid equilibrium is given by eq. 38 with the assumption of near constant relative volatilities.

$$\frac{\mathrm{d}H_B}{\mathrm{d}t} = -D(t) \tag{34}$$

$$\frac{\mathrm{d}\boldsymbol{X}_B}{\mathrm{d}t} = \frac{1}{H_B}[V(\boldsymbol{X}_N - \boldsymbol{Y}_B) + D(\boldsymbol{X}_B - \boldsymbol{X}_N)] \tag{35}$$

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{X}_n}{\mathrm{d}t} &= \frac{1}{H_n}[V(\boldsymbol{X}_{n-1} - \boldsymbol{X}_n) + \\ &+ D(\boldsymbol{X}_n - \boldsymbol{X}_{n-1}) + V(\boldsymbol{Y}_{n-1} - \boldsymbol{Y}_n)]\end{aligned} \tag{36}$$

$$\frac{\mathrm{d}\boldsymbol{X}_D}{\mathrm{d}t} = \frac{1}{H_D}[V(\boldsymbol{Y}_1 - \boldsymbol{X}_D)] \tag{37}$$

$$Y_i = \frac{\alpha_i X_i}{1 + \sum_{j=1}^{n-1}(\alpha_j - 1)X_j} \quad i = 1,\ldots,n-1 \tag{38}$$

The corresponding neural model is given by eqs. 39 – 41

$$\frac{\mathrm{d}X_B}{\mathrm{d}t} = f^{NN}(X_B, X_D, H_B, D) \tag{39}$$

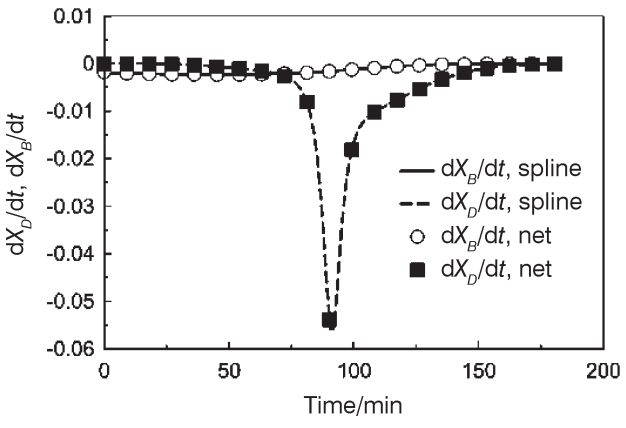
$$\frac{\mathrm{d}X_D}{\mathrm{d}t} = f^{NN}(X_B, X_D, H_B, D) \tag{40}$$

$$\frac{\mathrm{d}H_B}{\mathrm{d}t} = -D(t) \tag{41}$$

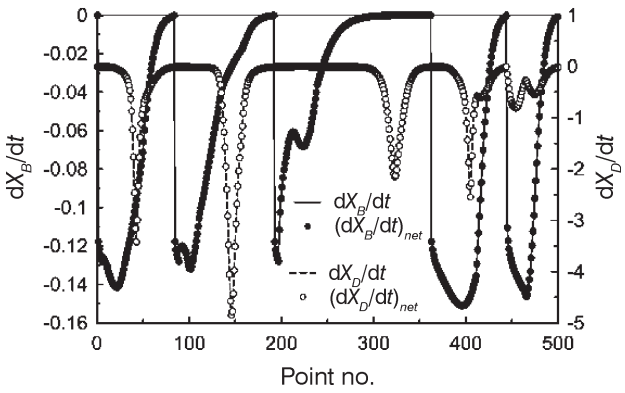
Model training has been done using a data set obtained by several simulated runs on different constant values of distillate flow. The data collected for top and bottom composition was then interpolated by cubic spline and derivatives were calculated for all points generated by spline interpolation. Fig. 13 shows interpolated data, derivation data, and corresponding network approximation for one batch run.

In order to cover the complete process range of interest, sets of batch runs are performed, and training and validation sets are formed. Because data gathered in this way belongs to different time series, poor fit or overfit inside particular runs can be prevented by adjusting the number of spline interpolated points. This process can be easily controlled graphically by plotting data against time. A greater problem is ensuring the quality of network interpolation of runs that belong to the time series of data not used in training. This was done by forming the validation sets from batch runs not included in the training sets (i.e. obtained from batches with distillate flows not used for training). As long as the results were unsatisfactory, the training set was increased by performing new batch runs by simulation using the physical model. Figs. 14 and 15 show the final results of training. Fig. 14 shows network training results on the training set, and Fig. 15 on the validation set.

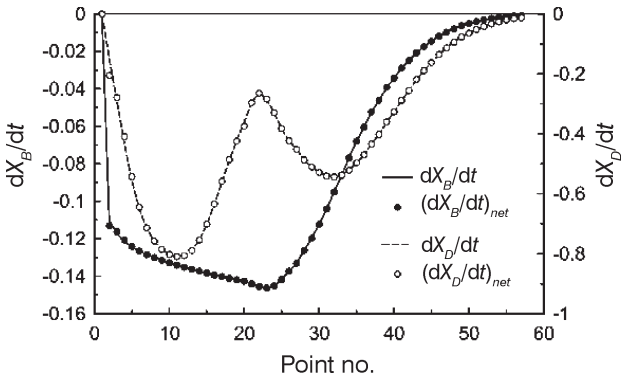
The developed neural model is used to perform the task of optimizing the distillate concentration during the batch run. If methanol concentration in the distillate is constant and sufficiently high, the regener-



F i g . 1 3 – *Learning of spline approximated derivative of top and bottom methanol concentration during distillation operation*



F i g . 1 4 – *Neural network approximation of distillation training set*



F i g . 1 5 – *Neural network approximation of distillation validation set*

ated solvent can be used directly in the production. For this purpose, the optimizing criteria is given by eq. 42, i.e. the optimal process is carried out if, during the run, the concentration of methanol as a product is kept constant, and in the end the desired quantity of methanol-free wastewater remains in the reactor.

$$J_1 = [H_f(t_f) - H_f^*]^2 + \int_0^{t_f} [X_i(t) - X_i^*]^2 \mathrm{d}t \quad (42)$$

Optimization was performed using the neural model with ten distillate flow segments. Fig. 16 shows the calculated optimal distillate flow that keeps methanol concentration at 99.5 % and 95.0 % respectively during the batch distillation operation.

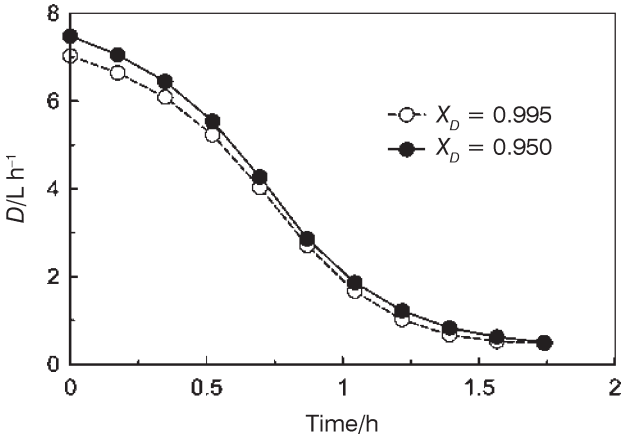


Fig. 16 – *Optimal distillate flow calculated with IDP method with goal of keeping the top concentration of methanol in distillate at two fixed values (X_D = 99.5 and 99.0)*

Generic neural model control was tested on oxidation operation. Considering the fact that hydrogen peroxide oxidizes a large number of unknown components present in the wastewater, the formal kinetic relation based on peroxide concentration only is developed (eq. 43)

$$r_A = k \cdot C_A^{aT+b} \quad (43)$$

Based on the kinetic model, the material and energy balance of the process is given by eqs. 44 – 46:

$$\frac{\mathrm{d}C_A}{\mathrm{d}t} = \frac{1}{V_r}\left[F_{in}C_A^{(in)} - V_r \cdot k \cdot C_A^{aT+b} - C_A\frac{\mathrm{d}V_r}{\mathrm{d}t}\right] \quad (44)$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{1}{V_r \cdot \rho \cdot C_{pA}}[F_{in}\rho_{in}C_{pA}(T_{in} - T_{ref}) + \\ + k \cdot V_r \cdot C_A \cdot \Delta H - U \cdot A \cdot (T - T_j)] - \frac{T}{V}\frac{\mathrm{d}V_r}{\mathrm{d}t} \quad (45)$$

$$\frac{\mathrm{d}T_j}{\mathrm{d}t} = F_{cm}\frac{\rho_{cm}}{m_{cm}}(T_{cm} - T_j) + \frac{U \cdot A}{C_{pcm}m_{cm}}(T - T_j) \quad (46)$$

The neural network equivalents for the model described by eqs. 44 – 46 is given by eqs. 47 – 49

$$\frac{\mathrm{d}C_A}{\mathrm{d}t} = f^{NN}(F_{in}, C_A, T, V_r) \quad (47)$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = f^{NN}(F_{in}, C_A, T, T_j, V_r) \quad (48)$$

$$\frac{\mathrm{d}T_j}{\mathrm{d}t} = f^{NN}(T_j, T, T_{cm}) \quad (49)$$

Networks 47 – 49 were trained in open loop process setup using data generated by process model (eqs. 44 – 46) by alternating sinusoidal control functions and random step functions (with inlet peroxide flow and thermostat temperature as inputs) in order to capture complete possible dynamic ranges of variables. The concentration network was the most difficult to train (eq. 47) due to instant changes in concentration in the reactor directly following input flow change. As for distillation, the total data set was divided into training and validation sets. Fig. 17 shows the results of concentration network simulation compared with complete training set, while Fig. 18 shows the same for the validation set.

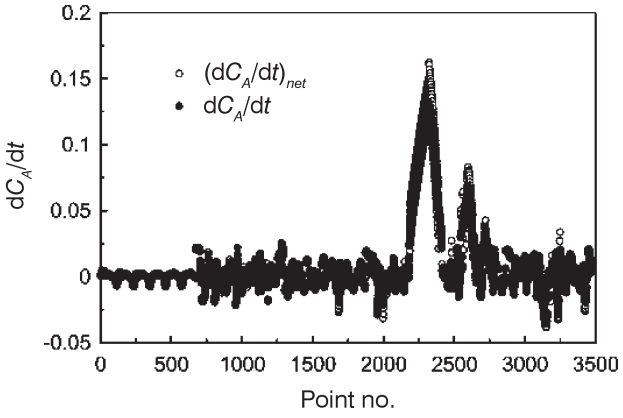


Fig. 17 – *Concentration network approximations in comparison with training set*

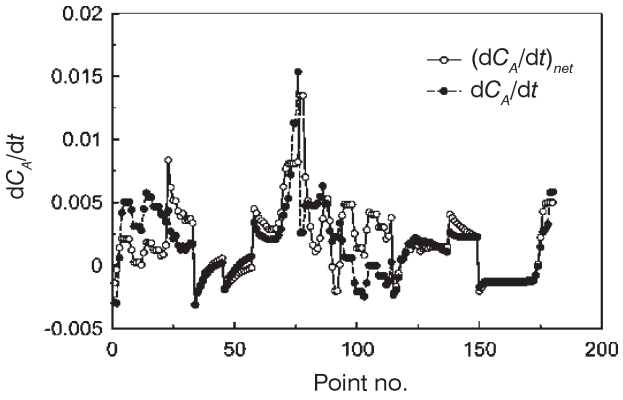


Fig. 18 – *Concentration network approximation in comparison with validation set*

After the training, the network models are used for generic neural model control testing. For this purpose, networks 47 and 49 were partially inverted as in eq. 29 in order to obtain explicit relations for control inputs F_{in} and T_{cm}. The comparison between classical PI cascade control of temperature and resulting GNMC control calculated by simulation is shown in Fig. 19.

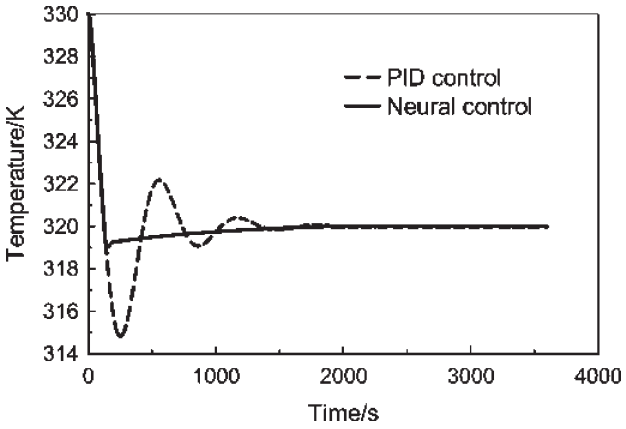


Fig. 19 – *Comparison of reactor temperature control during oxidation reaction with classical PID control and using generic neural model control*

Discussion

Besides the method here proposed, the task of optimization and batch process improvement can be achieved by direct recipe manipulation with the purpose of optimization (Verwater-Lukszo, Šel *et al.)*.[31,32] Verwater-Lukszo proposed a method for direct recipe optimization called FRIS (Flexible recipe improvement system) based on recipe adaptation. This adaptation is practically accomplished by the development and application of recipe adaptation set. The recipe adaptation set is prepared by utilizing combination of design of experiments, modeling and experimental optimization.

The main practical difference between recipe modification and the method introducing neural process abstraction layer here proposed is that the latter is tailored specifically to be in accordance with the quality by design approach. Since QbD approach allows free changes in the process as long as they do not move the process state space outside the boundaries of design space, no change in batch documentation is necessary from run to run. The neural network model works on process control inputs (manipulated variables) only, which means that batch data log is sufficient to document the differences between batch runs.

The necessary requirement for application of the proposed method is to build the recipe in the way described in this work. While this modification does not affect recipe flexibility, a problem may arise when attempting to apply this method to some batch plant with a different method of recipe building (ISA S88.01 allows and expects high flexibility in recipe formulation). The proposed method can still be applied, but change in recipe formulation may change the way the hybrid process model is built, so instead of hybrid automata, some other hybrid system model may be more appropriate. Consequently, changes in neural model structure and training may be necessary.

Training of neural network and its usage are the most important issues in application of the proposed method. Most of the difficulties arise from the necessity to approximate dynamical nonlinear model with neural network. In literature, there are many different methods developed involving recurrent (and for this reason either discrete or continuously dynamical) networks training. Unfortunately, most dynamical networks suffer from one or both common problems – complex and hardly convergent methods for training, as well as so-called "vanishing gradient problem" (gradually forgetting of older data dynamics behavior while learning from the more recent data set). For this reason, continuous state space representation of the neural network is used. Use of the spline smoothing and derivation of resulting polynomials enables overcoming the typical abovementioned problems during learning. One problem should nevertheless be taken into account – the problem of network regularization. It is well known that the main problem in using any multivariable nonparametric regression is the problem of data overfitting. Overfitting usually occurs when an applied network is able to learn more complex information than actually needed for a given data set (too many neurons or too many hidden layers in the network). When approximation of the multivariable function is performed, it is very difficult to predict possible overfitting. The simplest way to deal with this problem is to divide the training set into learning and verification sets. If verification fails, it is possible to resort to premature stopping of numerical minimization (learning algorithm) or repeating the learning with a less complex neural network. Things are somewhat different in the case of trajectory learning. The task of the neural network in this case is to approximate the trajectories presented as the training set, as well as to interpolate correctly between the trajectories, i.e. approximate correctly dynamical behavior of the system. Learning the trajectories in the training set may be accomplished by increasing the number of learning points after smoothing spline interpolation has been performed. Approximation quality (smoothness of trajectories) can be easily tested by plotting. On the other hand, interpolating between

trajectories is much more difficult to achieve. In this work, it was attained by dividing the smoothed trajectory data into two sets. One set was used for learning and the other for verification. If verification had failed, the network would be retrained with combined sets and another verification set would be produced from plant data. This procedure was repeated until verification passed. While this method is operational, its drawback is that sufficiently large data sets must be available. In the case of insufficient data, the alternative would be the application of the leave-one-out method as described by Specht[33–35] (treating trajectory as data point in static function approximation), but so far there is not enough experience to predict applicability of this method.

While in this work only neural models are used, due to the desire to achieve maximum possible generality, it is always advantageous to incorporate additional existing process knowledge into a developed model. If the use of a full thermodynamically-based model is impractical, there are two main approaches for incorporation of knowledge not accessible by the neural model alone.

The first method involves development of hybrid models which include some parts of the physical model. For example, Hosen *et al.*[36] used hybrid neural model to control polystyrene batch polymerization reaction where the first principle kinetic model was used as a part of the overall neural model.

The second method involves the combining of AI methods in order to incorporate different kinds of process knowledge offered by different algorithms. Especially interesting is the combination of neural and fuzzy systems due to favorable incorporation of symbolic knowledge into numeric neural model. Examples of such neuro-fuzzy systems are given by Causa *et al.*[37] and Simon and Hungerbuhler.[38]

In practical implementation of a developed method it is also important to address the problem of software realization of neural network applied. It is obvious that software modules for data gathering and network training should be preferably placed on SCADA system or some separate computer on industrial LAN network. On the other hand, the network execution modules along with weight sets need to be in PLC or DCU system, because they have the logical role of a layer between IO modules and basic control of equipment modules. In the case of the PCS7 system, they can be implemented with the same tool used for basic control programming – continuous function chart (CFC) tool. If PLC does not have sufficient processing power or a lower programming language is used for software development, a possible solution is to use a separate processor for network execution.

Conclusion

The presented method of installing the neural network layer between the process and basic control level in a recipe-controlled batch plant, is shown to have potential in dealing with the task of constant gradual improvement by movement of control space (parameter operating range space) inside the design space.

In future work, attention should be paid to further development of methods of hybrid neural modeling that would allow implementation of neural process layer to a broader class of recipe structures. In addition, dynamic neural network training methods with reduced data set should also be a topic of interest.

List of symbols

H – liquid holdup, mol
D – distillate flow, mol s^{-1}
$\boldsymbol{X}$ – molar fraction in liquid phase
$\boldsymbol{Y}$ – molar fraction in vapor phase
V – vapor flow, mol s^{-1}
t – time, s
α – relative volatility
C_A – molar concentration, mol L^{-1}
F – flow, L h^{-1}
V_r – reactor volume, L
T – temperature, K (°C)
ρ – density, kg m^{-3}
ΔH – reaction enthalpy, J mol^{-1}
A – area, m^2
U – heat transfer coefficient, W m^{-2} K^{-1}
C_p – heat capacity, J mol^{-1}
m – mass, kg

Indices

B – bottom
D – distillate
n – theoretical stage number
f – final
r – reactor
j – jacket
in – inlet
cm – cooling medium
NN – neural network

Literature

1. ANSI/ISA-95.00.01–2000, Enterprise-Control System Integration, Part 1: Models and Terminology.
2. ANSI/ISA S88.01–1995(2011), Batch Control Part 1: Models and Terminology.

3. *Lepore, J., Spavins, J.,* Journal of Pharmaceutical Innovation,. **3** (2008) 79.
4. *Davies, B., Ellis, S.,* Pharmaceutical Technology Europe, **17** (2005) 17, 22.
5. *Sanchez, A., Para, L., Baird, R., Macchietto, S.,* ISA Transactions **42** (2003) 401.
6. *Moor, T., Raisch, J.,* Supervisory Control of Hybrid Systems In: Behavioural Framework. Systems and Control Letters, Elsevier, New York, 1999, 157–166.
7. *Potočnik, B., Bemporad, A., Torrisi, F., Mušič, G., Zupančič, B.,* Control Engineering Practice **12** (2004) 1127.
8. *Goebel, R., Sanfelice, R., Teel, A.,* Hybrid Dynamical Systems, Princeton University Press New Yersey, 2012.
9. *Lennartson, B., Tittus, M., Egardt., B., Pettersson, S.,* IEEE Control Systems **16** (1996) 45.
10. *Barton, P., Banga, J., Galan, S.,* Computers and Chemical Engineering **24** (2000) 2171.
11. *Fabre, F., Hetreux, G., Le Lann, J., Zarate, P.,* Computers and Chemical Engineering **35** (2011) 2098.
12. *Haykin, S.,* Neural networks and learning machine, Prentice Hall, New York, 2009.
13. *Masters, T.,* Advanced algorithms for neural networks, John Wiley, New York, 1995.
14. *Hrycej, T.,* Neurocontrol, John Wiley, New York, 1997.
15. *Paengjuntuek, W., Thanasinthana, L., Arpornwichanop, A.,* Neural Neurocomputing **83** (2012) 158.
16. *De Jesus, O., Hagan, M.,* IEEE Transaction on Neural Network **18** (2007) 15.
17. *Werbos, P.*, Proceedings of IEEE **78** (1990) 1550.
18. *Williams, R., Zipser, D.,* Neural Computation **1** (1989) 270.
19. *Nerrand, O., Roussel-Ragot, P., Urbani, D., Personnaz, L., Dreyfus, G.,* IEEE Trans. on Neural Networks **5** (1994) 178.
20. *Gosak, D., Vampola, M.,* Generic Neural Model Control of Batch Distillation Process In: Advances in Process Control 6 , *Love, J.* (Ed.). Rugby: ICHEME UK, 2001. 137 – 144.
21. *Alur, R., Henzinger, T. A., Lafferriere, G., Pappas, G.,* Proceedings of the IEEE **88** (2000) 971.
22. *Le Sceller, L., Letellier, C., Gouestbed*, *G*., Physics Letters A **211** (1996) 211.
23. *De Boor, C.,* Practical guide to splines, Springer, New York, 2001.
24. *Chen, H. H., Manry, M. T., Chandrasekaran, H.,* Neurocomputing **25** (1–3) (1999) 55.
25. *Grewal, M., Andrews, A.,* Kalman filtering theory and practice using MATLAB, John Wiley and Sons, New York, 2001.
26. *Abonyi, J., Madar, J., Szeifert, F.,* Combining first principles models and neural networks for generic model control In: Soft Computing in Industrial Applications – Recent Advances, *Roy, R., Koppen, M., Ovaska, S., Furuhashi, T., Homann, F.* (Eds.), Springer Engineering Series, 2001, 111–122.
27. *Lee, P., Sullivan, G.,* Computers and Chemical Engineering **12** (6) (1988) 573.
28. *Bojkov, B., Luus, R.,* Industrial & Engineering Chemistry Research **31** (1992) 1308.
29. *Schroder, A., Mendes, M.,* Computers and Chemical Engineering Supplement (1999) S491.
30. *Vampola, M., Gosak, D., Šoštarec, A, Pavličić, D., Vraneš, N., Hranilović, M*., Development of the Batch Reaction Control System Using the ISA S88 Batch Control Standard, In: Computers in Technical Systems and Intelligent Systems, *Budin, L.* (Ed.). Rijeka MIPRO, 2002.
31. *Verwater-Lukszo, Z.,* Computers in Industry **36** (1998) 279.
32. *Šel, D., Hvala, N., Strmčnik, S., Milanič, S., Šuk-Lubej, B.,* Control Engineering Practice **7** (1999) 1191.
33. *Larsen, J., Svarer, C., Nonboe Andersen, L., Hansen, L. K.,* Adaptive Regularization in Neural Network Modeling In: *Orr, G. B., Müller, K*. (Eds.) Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science 1524, Germany: Springer-Verlag, 1998, 113–132.
34. *Specht, D.,* IEEE Transactions on Neural **2** (6) (1991).
35. *Boser, B., Guyon, I., Vapnik, V.,* A training algorithm for optimal margin classifiers In: 5th Annual Workshop on Computational Learning Theory, Pittsburgh, ACM, 1992, 144–152.
36. *Hosen, M., Husain, M., Mjalli, F.,* Control Engineering Practice **19** (2011) 454.
37. *Causa, J., Karer, G., Nunez, A., Saez, D., Škrjanc, I., Zupančič, B.,* Computers and Chemical Engineering **12** (2008) 3254.
38. *Simon, L., Hungerbuhler, K.,* Chemical Engineering Journal **157** (2010) 568.