Software Application for Adaptive Identification and Controller Tuning

Bc. Peter Mark Beneš

Supervisor: Doc. Ing. Ivo Bukovský Ph.D.

Abstract

This paper introduces a new software for adaptive identification and controller tuning, with the use of higher-order neural units and gradient descent based techniques (including back-propagation through time). The software allows the user to load real process data offline and to identify the plant or control loop as a whole. Furthermore the software experimentally investigates potentials for optimisation of the control loop response, via a non-linear adaptive state-feedback controller. The software is aimed as a quick tool for students, scholars and practitioners who wish to check potentials for optimisation of their control loop (utilising available process data and non-linear controller).

Keywords

software application, real-data based identification and control, real time recurrent learning, back-propagation through time, gradient descent, dynamic linear neural unit, dynamic quadratic neural unit, adaptive feedback controller, Neuro-Controller, SISO

1. Introduction

The work of my research together with the software application presented in this paper is motivated by the development of Linear and Quadratic Neural Units (LNUs) and (QNUs) respectively [1][2]. So far such neural units feature promising theoretical studies for adaptive identification and control [9]–[11] along with successful real implementation in the Automatic Control laboratory at CVUT [11] as recalled in Figure 1.

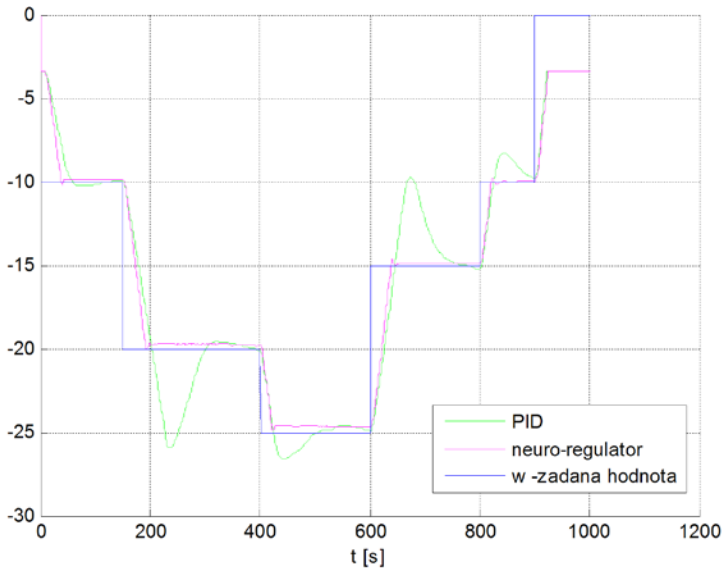

Figure 1: *Demonstration of Quadratic Neural Unit as a controller on the Bathyscaphe System (picture adopted from [11]), real implementation of QNU with gradient descent was superior to PID control.*

Here we can see that real implementation of the Neuro-controller [11] is indeed the most desirable controller for the given process data, following more closely to the desired behaviour of the system than the linearly limited PID controller. Given this, further motivation arises in applying and extending this algorithm for application to other real engineering processes.
The designed algorithms that I implemented, furthermore the software that I programmed, are based on the above algorithms and referenced works. It is an attempt to provide users with a more unified tool where real data can be loaded and the user can see what could be the potentials for further optimization of a control loop by this gradient descent based approaches [1]–[11].
Of course, there might be difference between simulation and real functioning of the implemented controller (e.g. as observed in [11]), nevertheless the developed software is aimed to indicate the potentials for optimisation of the process control.

2. Description of Implemented Algorithms

The background behind the adaptive control utilised in this software application is the well-known Gradient Descent (GD) method applied to dynamic adaptive models. The applied learning rule for dynamic neural units is based on incremental and batch training techniques. The incremental (sample-by-sample) adaptation is based on Real Time Recurrent Learning (RTRL) technique [3]. The batch adaptation is based on modification of Back-Propagation Through Time (BPTT) technique [4] implemented as a combination of RTRL with the famous Levenberg-Marquardt algorithm as shown in [8].

In the implemented software, both these learning techniques (RTRL and BPTT) can be independently used, or combined to adaptively identify a system, i.e. either a plant or even a whole control loop, and to adaptively tune the controller to demonstrate potentials for further optimisation of the control process.

So far, the whole application has been designed for considering only single-input single-output systems.

2.1 Gradient Descent Adaptation

The Gradient Descent algorithm is one of the most fundamental learning rules behind neural units such as LNU and QNU (e.g. [1][2]) used throughout the program. This method is suitable for both online and offline tuning of static and, more importantly for this paper, dynamic models. The essence of this algorithm is to learn the model of the plant, however it can also be utilised in tuning a controller such as the neuron type controller (Neuro-controller) [9]–[11].

Firstly we introduce the general form of the LNU respectively QNU, as expressed via the following polynomial forms;

$$y = \sum_{i=0}^{n} x_i w_i = w_0.x_1 + w_1.x_2 + \ldots + w_n.x_n = \mathbf{w.x} \tag{1}$$

where **w** is updatable vector of neural weights and **x** is vector of inputs in the case of a purely static model, or as here also previous outputs corresponding to a dynamic model. Similarly, the quadratic neural unit can be expressed as follows;

$$y = \sum_{i=0}^{n}\sum_{j=0}^{n} x_i x_j w_{i,j} = w_{0,0}x_0x_0 + w_{0,1}x_0x_1 + \ldots + w_{n,n}x_n^{\,2} = \mathbf{rowx.colW} \tag{2}$$

Adopting the long vector notation according to [6][7], QNU can be expressed as;

$$y = \begin{bmatrix} x_0 x_0 & x_0 x_1 & x_0 x_2 \ldots & x_n x_n \end{bmatrix} . \begin{bmatrix} w_{0,0} \\ w_{0,1} \\ \vdots \\ w_{n,n} \end{bmatrix} = \mathbf{rowx.colW} \, , \tag{3}$$

where **rowx** and **colW** are long-vector representations of the input vector and weight matrix of the quadratic neural unit in general. The goal behind this algorithm is adaptation of neural weights, this is the key behind the learning process of the model. This is achieved via modifications of the fundamental gradient descent formula for the LNU and QNU respectively as follows;

$$w_{i+1} = w_i + \mu . e(k) . \frac{\partial y(k)}{\partial w_i} \tag{4}$$

where μ represents the learning rate of the weight adaptation, $e(k)$ represents current error between real and calculated output. The final term $\frac{\partial y(k)}{\partial w_i}$ corresponds to the partial derivatives of the neural unit output, respective to each neural weight. For understanding in how this partial derivative is computed we shall derive it in the following section. As for the QNU, the Gradient Descent algorithm is as follows;

$$\mathbf{colW}(\mathrm{k}+1) = \mathbf{colW}(k) + \mu . e(k) . \frac{\partial y(k+1)}{\partial \mathbf{colW}} \tag{5}$$

2.2 Adaptation of Dynamic Linear Neural Units

If we say that our linear neural unit will be dynamic (with feedbacks from output), we can choose our vector **x** to feed previous samples of output to input, for example,

$$x = \begin{bmatrix} 1 \\ y[k] \\ y[k-1] \\ y[k-2] \\ u[k] \\ u[k-1] \end{bmatrix} \begin{matrix} \\ \Big\} ny \\ \\ \Big\} nu \end{matrix}$$

where *ny* and *nu* are numbers of corresponding components of input vector. In such case we will have 6 weights, corresponding to each of the elements of input vector x. We can thus expand equation (4) in the following manner.

$$\frac{\partial y(k+1)}{\partial w_i} = \frac{\partial}{\partial w_i} (w_0 + w_1 . y[k] + w_2 . y[k-1] + \ldots + w_5 . u[k-1]) \tag{6}$$

Thus for our dynamic model equation, all terms denoted as *y* are previous output values, the total amount of which is *ny*. These previous output values of *y*, were calculated using the same model equation at different sampling times of *k*, thus here too we have a function of the previous weights. We thus find that on differentiation of these individual terms we will utilise the product rule of differentiation, which yields the following matrix.

$$\frac{\partial y(k+1)}{\partial w_i} = \begin{bmatrix} \frac{\partial y(k+1)}{\partial w_o} \\ \frac{\partial y(k+1)}{\partial w_1} \\ \frac{\partial y(k+1)}{\partial w_2} \\ \frac{\partial y(k+1)}{\partial w_3} \\ \frac{\partial y(k+1)}{\partial w_4} \\ \frac{\partial y(k+1)}{\partial w_5} \end{bmatrix} = \begin{bmatrix} 1 & w_1.\frac{\partial y(k)}{\partial w_0} & w_2.\frac{\partial y(k-1)}{\partial w_0} & w_3.\frac{\partial y(k-2)}{\partial w_0} \\ y[k] & w_1.\frac{\partial y(k)}{\partial w_1} & w_2.\frac{\partial y(k-1)}{\partial w_1} & w_3.\frac{\partial y(k-2)}{\partial w_1} \\ y[k-1] & w_1.\frac{\partial y(k)}{\partial w_2} & w_2.\frac{\partial y(k-1)}{\partial w_2} & w_3.\frac{\partial y(k-2)}{\partial w_2} \\ y[k-2] & w_1.\frac{\partial y(k)}{\partial w_3} & w_2.\frac{\partial y(k-1)}{\partial w_3} & w_3.\frac{\partial y(k-2)}{\partial w_3} \\ u[k] & w_1.\frac{\partial y(k)}{\partial w_4} & w_2.\frac{\partial y(k-1)}{\partial w_4} & w_3.\frac{\partial y(k-2)}{\partial w_4} \\ u[k-1] & w_1.\frac{\partial y(k)}{\partial w_5} & w_2.\frac{\partial y(k-1)}{\partial w_5} & w_3.\frac{\partial y(k-2)}{\partial w_5} \end{bmatrix}, i = 0,1,2..5 \quad (7)$$

This derivation can thus be analogically applied to all other various other types of model polynomial equations. In the sense of the plant identification algorithms used in this software, it became apparent that a much simpler version of this derivation seemed to work better in the various tested data, where by $\frac{\partial y(k+1)}{\partial w_i} = \mathbf{x}$. This is perhaps due to the fact that all proceeding terms of this partial differentiation are recurrently trained variables, which seem to somewhat complicate the update of the weight training process. Another component of this algorithm, necessary to mention is the learning rate μ, this plays a key role in the adaptation algorithm. It essentially corresponds to the speed of learning for the neural unit, should μ be high, then so too is the rate of learning for the neural model. Analogically setting μ to a smaller value dictates slower learning for the model. This can be advantageous as the model is allowed to process the learning more effectively ie; analogical to humans where the longer time is given to learn, the more easily can a human remember the information. Thus through every adaptation step *k*, the model is trained to represent the behaviour of the process for which its data was provided.

Often we may find that for different process data it is better to normalise the learning rate due to problems associated with instability during the learning of the neural units. In theory [6], the fundamental normalisation of the learning rate is given as follows;

$$\eta = \frac{\mu}{\|\mathbf{rowx}(k)\|_2^2 + \varepsilon} \quad (8)$$

where $\|\mathbf{rowx}(k)\|_2^2$ is the Euclidean norm of the vector **rowx**, furthermore ε represents a normalisation constant, which is updatable over adaptation step k. However in this software it was apparent that a more simplified version of this learning rate normalisation can be used as follows;

$$\eta = \frac{\mu}{\mathbf{x}(k)\mathbf{x}(k)^T + 1} \quad (9)$$

2.3 Adaptation of Dynamic Quadratic Neural Units

Adopting the long term notation for the QNU training [2][6][7], the above equation (3) defines the QNU. For further definition as per equation (12) we must also distinguish the following;

$$\mathbf{colx} = \mathbf{rowx}^{T} \tag{10}$$

When looking at the already introduced equation of the gradient descent algorithm as per equation (5), here we see that in this case the partial derivative is in terms of **colW**, In general this partial derivative would thus be represented via the following matrix term;

$$\frac{\partial y(k+1)}{\partial \mathbf{colW}} = \mathbf{colW}.\begin{bmatrix} \frac{\partial \mathbf{rowx}}{\partial w_{0,0}} \\ \frac{\partial \mathbf{rowx}}{\partial w_{0,1}} \\ \vdots \\ \frac{\partial \mathbf{rowx}}{\partial w_{n,n}} \end{bmatrix} = \mathbf{colW}.\begin{bmatrix} \frac{\partial x_o^{\,2}}{\partial w_{0,0}} & \frac{\partial (x_o.x_1)}{\partial w_{0,0}} & \cdots & \frac{\partial x_n^{\,2}}{\partial w_{0,0}} \\ \frac{\partial x_o^{\,2}}{\partial w_{0,1}} & \frac{\partial (x_o.x_1)}{\partial w_{0,2}} & \cdots & \frac{\partial x_o^{\,2}}{\partial w_{0,1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial x_o^{\,2}}{\partial w_{n,n}} & \frac{\partial (x_o.x_1)}{\partial w_{n,n}} & \cdots & \frac{\partial x_n^{\,2}}{\partial w_{n,n}} \end{bmatrix} \tag{11}$$

In addition to this, the learning rate μ may be replaced via a normalized learning rate in aid of better stability in certain applications of this model, as follows;

$$\eta = \frac{\mu}{\mathbf{colx}(\mathrm{k})\mathbf{colx}(\mathrm{k})^{T} + 1} \tag{12}$$

2.4 Extension of Gradient Descent Method on the Neuro-Controller

To understand the mechanics behind the gradient descent algorithm, as applied to the Neuro-Controller setup, we should first consider the control scheme (loop) as follows;

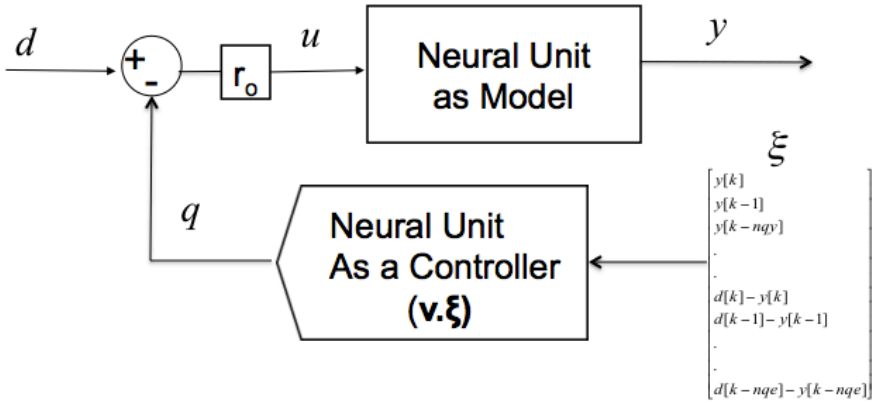

Figure 2: *Control Scheme of the Neuro-Controller.*

where r_o is adaptable proportional gain. In Figure 2 as above, we see two main neural unit blocks denoted **w.x** and **v. ξ**. In this control scheme block **w.x** refers to the plant identifier (used to similarly to learn the behaviour for the given process data) and **v. ξ**. is thus the

Neuro-Controller used to manipulate the newly feed input into the neural model for control. In this case the neural weight update would be dictated via the gradient descent algorithm in the following way;

$$v_{i+1} = v_i + \mu.ereg(k).\frac{\partial y(k)}{\partial v_i} \tag{13}$$

where v_i are adaptable neural weights of the Neuro-controller and *ereg*(*k*) is the error between the desired value of the plant (denoted *d*) and the real plant output value at sample *k*. The most crucial component behind equation (13) is hidden within the partial derivative $\frac{\partial y(k)}{\partial v_i}$, via application of the chain rule we may derive the final form of this derivative for use in the programming algorithm as;

$$\frac{\partial y}{\partial v} = \mathbf{w}.\frac{\partial x}{\partial v} = -\mathbf{w}.\frac{\partial q}{\partial v} \tag{14}$$

where $\frac{\partial q}{\partial v}$, corresponds to the partial derivatives from the Neuro-Controller equation with respect to the updatable weights **v**. In order to understand computation of this partial derivative we must define matrix ξ**.** For example as represented in Figure 2, we may define as;

$$\xi = \begin{bmatrix} y[k] \\ y[k-1] \\ y[k-2] \\ d[k]-y[k] \\ d[k-1]-y[k-1] \\ d[k-2]-y[k-2] \end{bmatrix} \begin{matrix} \} \text{nqy} \\ \\ \\ \} \text{nqe} \end{matrix} \tag{15}$$

Thus for this case, the partial derivative with respect to the updatable Neuro-Controller weights v_i, yields that in fact $\frac{\partial q}{\partial v} = \xi$. Analogically this result was applied in the presented software, where the length of previous results of *y* is adaptable via the edit entitled "*nqy*" within the Neuro-Controller panel. Similarly too the length of previous samples for the difference of desired to model output (d-y) is adaptable via the parameter *nqe*.

2.5 Real Time Recurrent Learning, Back-Propagation Through Time

Throughout this software there are two key methods of weight training or learning utilised. These methods are known as Real Time Recurrent Learning [3] or shortly RTRL and Back-Propagation Through Time [4], shortly BPTT. So far the gradient descent algorithm was introduced above as per equation (4) for DLNU and equation (5) for DQNU. These equations are in essence the Real Time Recurrent Learning, however what is key to distinguish is that this only applies for dynamic adaptive models, where adaptation is achieved over sample by sample.

Thus another approach for adaptive learning is to in fact train, rather than over each sample, over a series of runs or epochs of the neural algorithm. The advantage of this is evident for instance in noisy data. Here rather than learning with noise over sample by sample as in the RTRL method, we can train over each run or "Batch" where by the main governing law

interests in training of the input and output for each run. This is thus referred too as Batch training or furthermore Back-Propagation Through Time (BPTT). To understand this method in more depth we must introduce a very key equation according to Levenberg-Marquardt algorithm as follows

$$\Delta \mathbf{w} = (\mathbf{J}.\mathbf{J}^T + \frac{1}{\mu}.\mathbf{J})^{-1}.\mathbf{J}.\mathbf{e}. \tag{16}$$

Here the weight update algorithm would thus be calculated as w=w+Δw. In equation (16) the term I, is simply an identity matrix. **J** denotes the Jacobian matrix of derivatives for the model polynomial equation. In our case for this software the partial derivative of the model polynomial with respect to the adaptable weights **w** ie; $\frac{\partial y(k)}{\partial w_i}$, simply equals to **x** for DLNU or in the sense of DQNU, **colx.** Thus an extension to equation (16) for batch training (BPTT) would be for LNU as follows

$$\Delta \mathbf{w} = (\mathbf{x}.\mathbf{x}^T + \frac{1}{\mu}.\mathbf{I})^{-1}.\mathbf{x}.\mathbf{e}$$

and for QNU as follows;

$$\Delta \mathbf{w} = (\mathbf{colx}.\mathbf{colx}^T + \frac{1}{\mu}.\mathbf{I})^{-1}.\mathbf{colx}.\mathbf{e} \tag{17}$$

3. Results

3.1 Designed Software

The presented software in this paper, was created via Python 2.6.2 programming language, with use of the graphical interface library wxPython. The Interface features two distinctive tabs on the main window, which refer to control of a plant (process) and furthermore control for a Plant as part of a proportional control loop (P-Loop). It should be stressed that the version presented in this paper is a prototype, with further developments to be added in the close future.

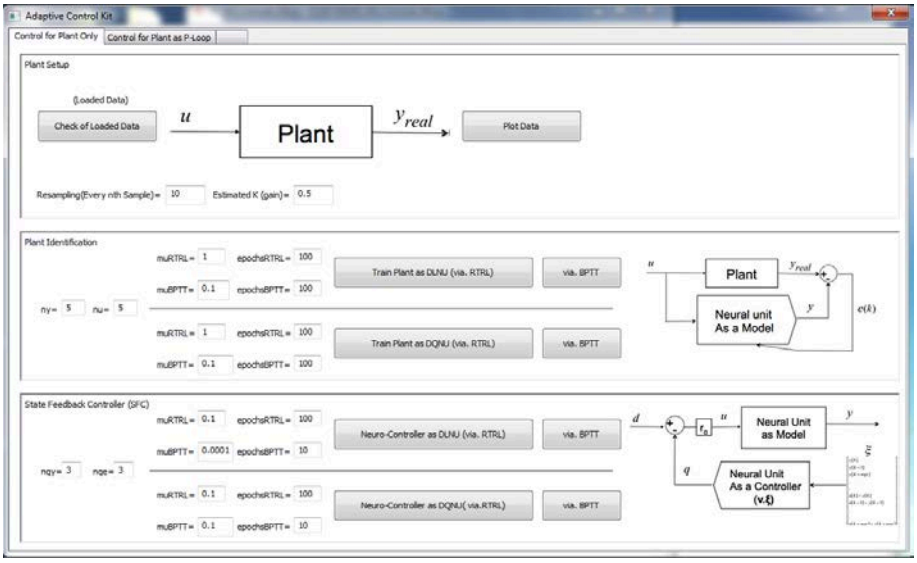

Figure 3: *Main Interface of Software – Option for only plant data.*

The above figure (Figure 3) shows the current layout within the “Control of Plant Only” window. Here the frame is split to 3 distinct panels. The first allows the user to check their uploaded process data is in the path of the program via the “Check Loaded Data” button. Following this is the “Plot Data” button, enabling the user to see a visualisation of their uploaded process data. After the visualisation of the data is created, the user may require resampling of the data, for the purpose of control. Thus an edit titled “Resampling” is featured on the bottom of this first panel.

The following panels of this program are divided into “Plant Identification” and “State Feedback Controller (Neuro-Controller)”. Here the user must fill in the learning rate for usage of gradient descent algorithm and furthermore epochs. As the values of the learning rate and number of epochs differ between the RTRL and BPTT methods, separate edits are placed for each within each panel. There is also a feature for the user to define the length of the variables used in the model polynomial equations for the respective controller, all of which is explained under section 2 in “Description of Implemented Algorithms”.

Control may be calculated for the following options : DLNU Plant Identification with DLNU Neuro-Controller, DLNU Plant Identification with DQNU Neuro-Controller, DQNU Plant Identification with DLNU Neuro-Controller and DQNU Plant Identification with DQNU Neuro-Controller. This thus gives the user the option to compare behaviour of the different methods, furthermore allowing the user to tune for most optimal control to their process data.

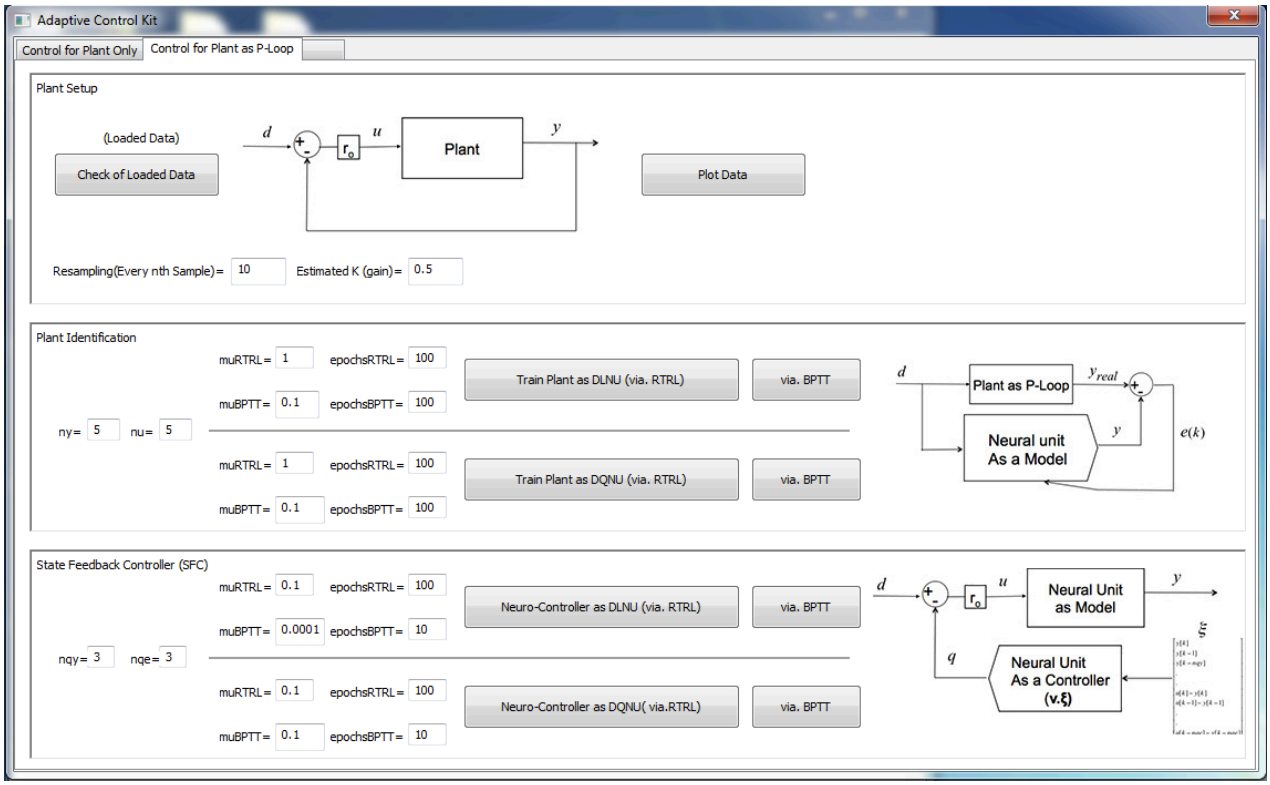

***Figure 4**: Main Interface of Software – Option for data measured from control P-Loop.*

An extension to this software is seen in the second tab under Figure 4. Here the Plant may be introduced as part of a Proportionally Controlled loop shortly (P-Loop). The functions of this panel are analogical to above. This panel proves good for such scheme as presented in the Bathyscaphe results of the proceeding section.

3.2 Simulations Using the Designed Software

The results presented in this section are mainly based on a theoretical second-order plant, with a pulsating signal for Input. However this software was also tested on a real process data, which was that of the Bathyscaphe system. In the sense of the theoretical process the following transfer function defined our example Plant;

$$G(s) = \frac{s + 0.1}{3s^2 + 2s + 1} \tag{18}$$

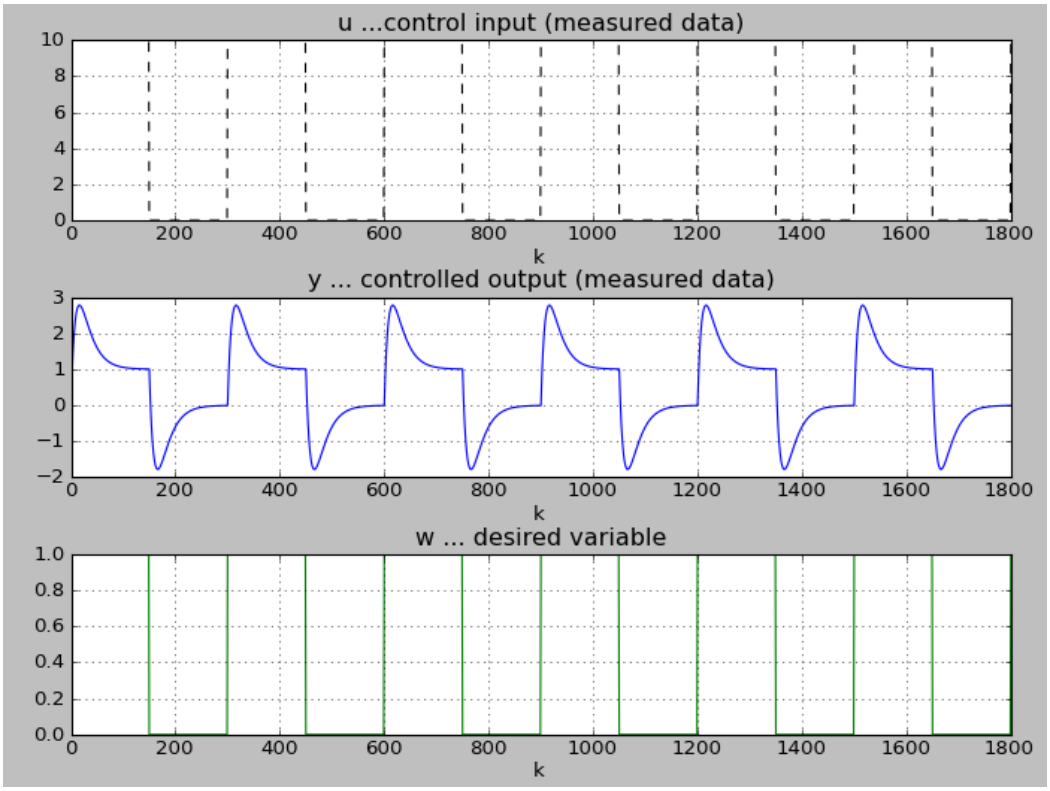

Figure 5: *Preview of loaded (process) data in a program (Data should be in same folder as the Program, thus Check Loaded Data button may be utilised)*

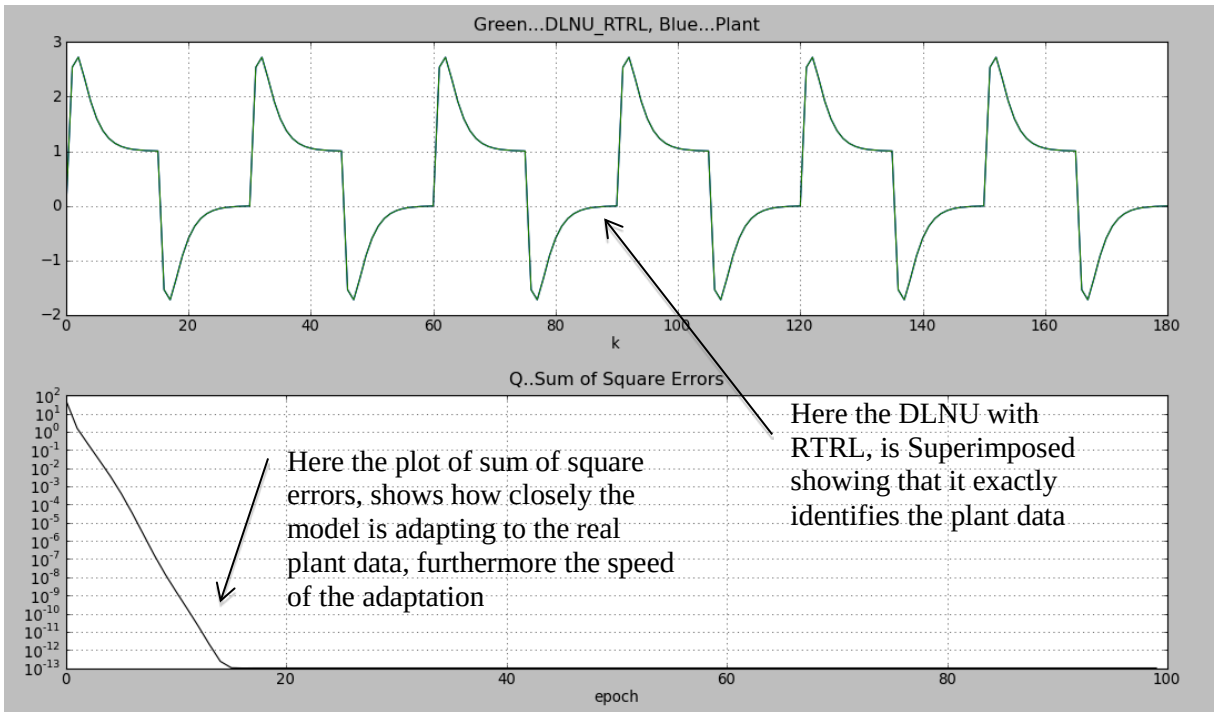

Figure 6: *Plant Identification Using DLNU with RTRL Training mu=1, epochs 100, For ny=5 and nu=5*

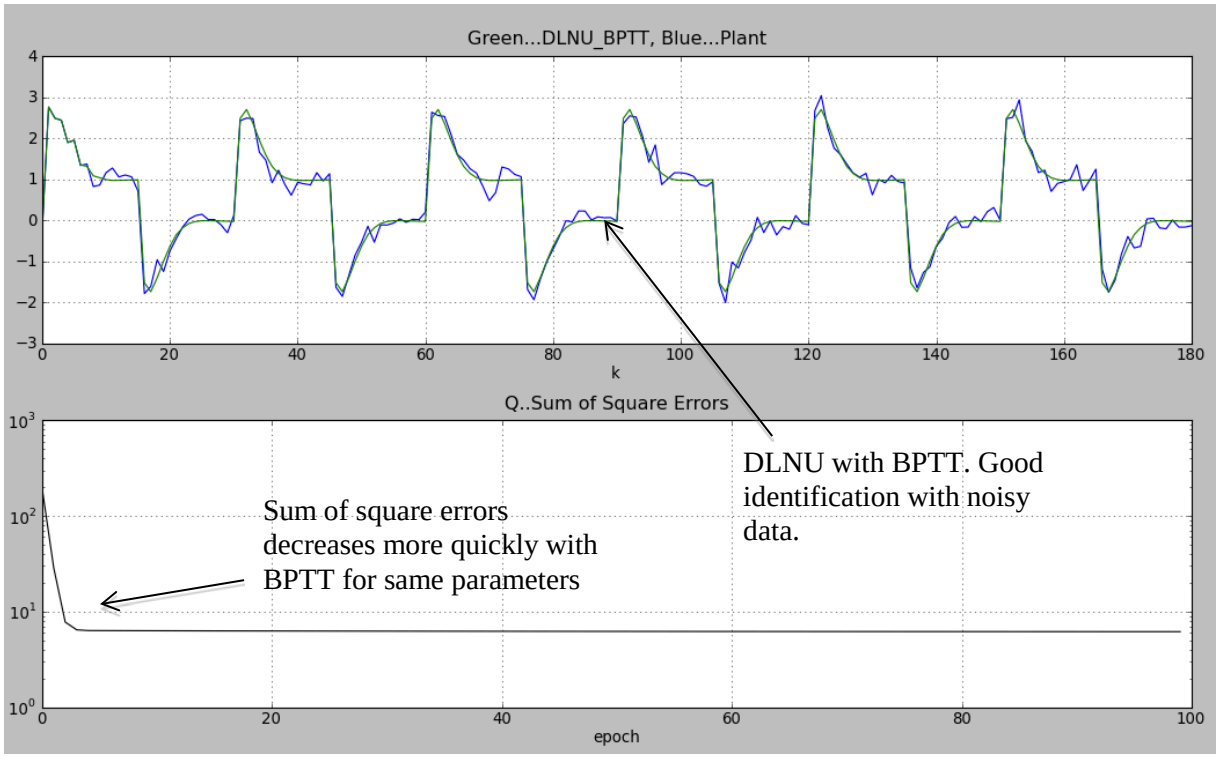

Figure 7: *Plant Identification Using DLNU with BPTT Training mu=1, epochs 100, For ny=5 and nu=5*

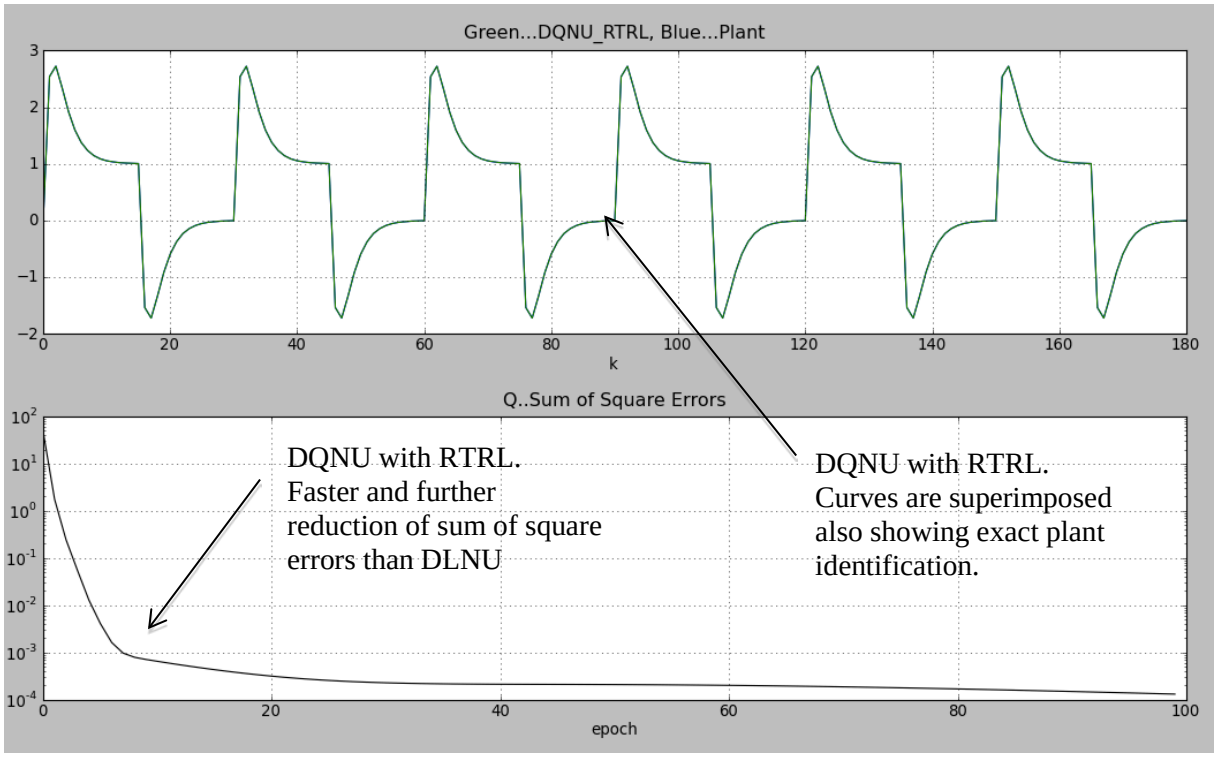

Figure 8: *Plant Identification Using DQNU with RTRL Training mu=1, epochs 100, For ny=5 and nu=5*

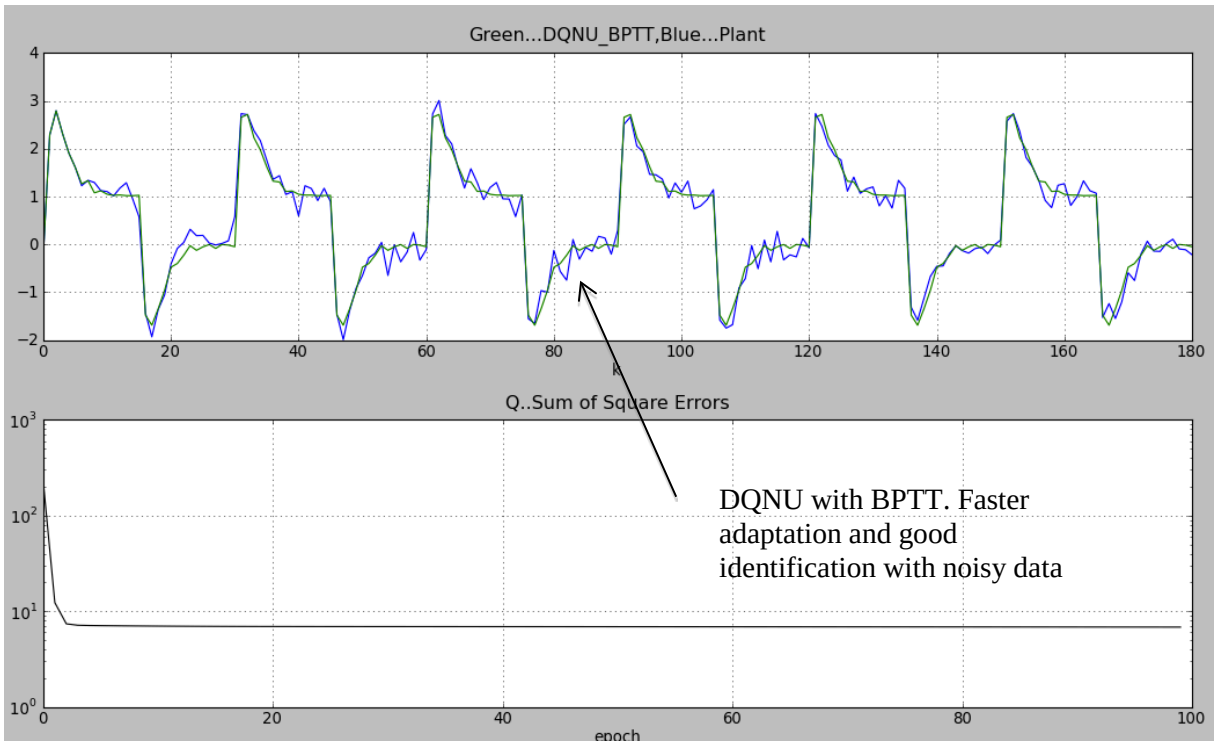

Figure 9: *Plant Identification Using DQNU with BPTT Training mu=1, epochs 100, For ny=5 and nu=5*

Results of the Neuro-Controller Given the Previously Identified Plant (Process)

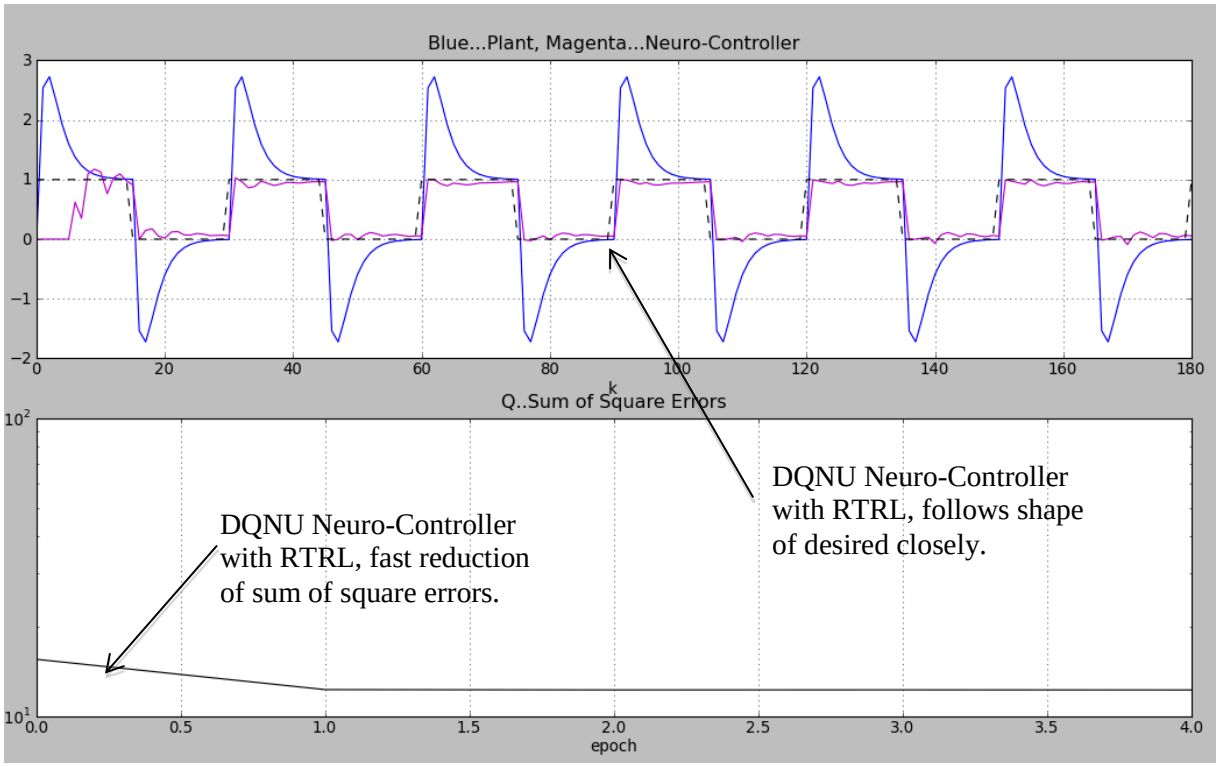

Figure 10: *Neuro-Controller Using DQNU with RTRL Training mu=0.2, epochs 5, nqy=4 , nqe=4*

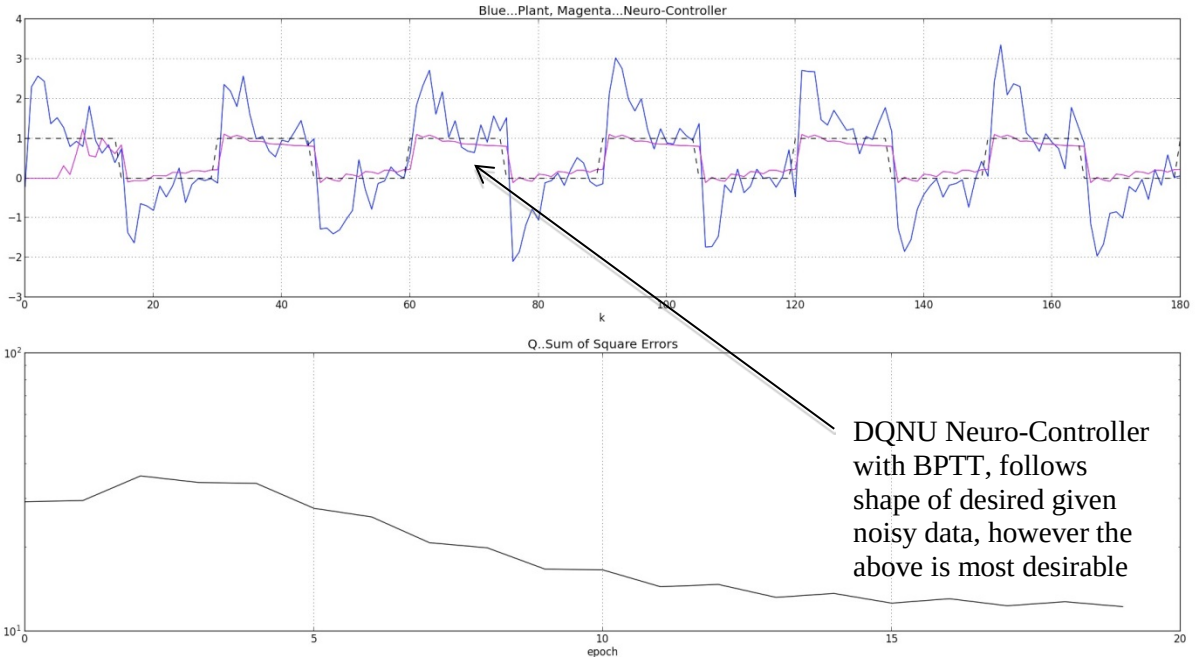

Figure 11: *Neuro-Controller Using DQNU with BPTT Training mu=0.1, epochs 100, nqy=4, nqe=4*

Application on Bathyscaphe Data

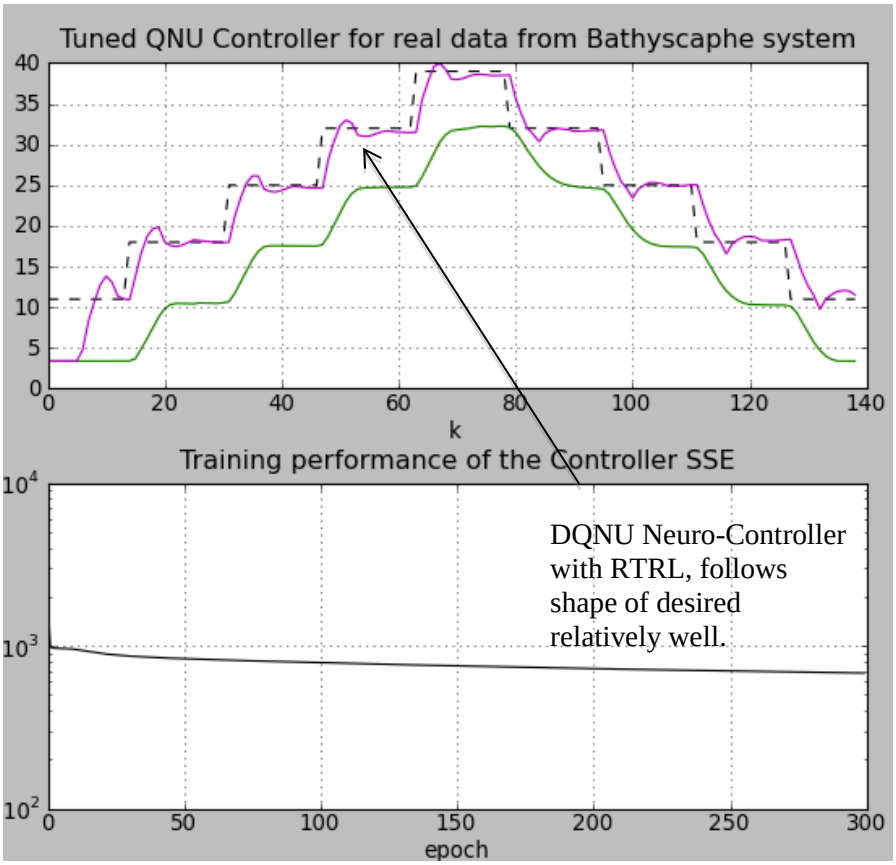

Figure 12: *Bathyscaphe Data- DQNU_RTRL Neuro-Controller Using DQNU_RTRL Plant Identification (Control of Plant as P-Loop tab used)*

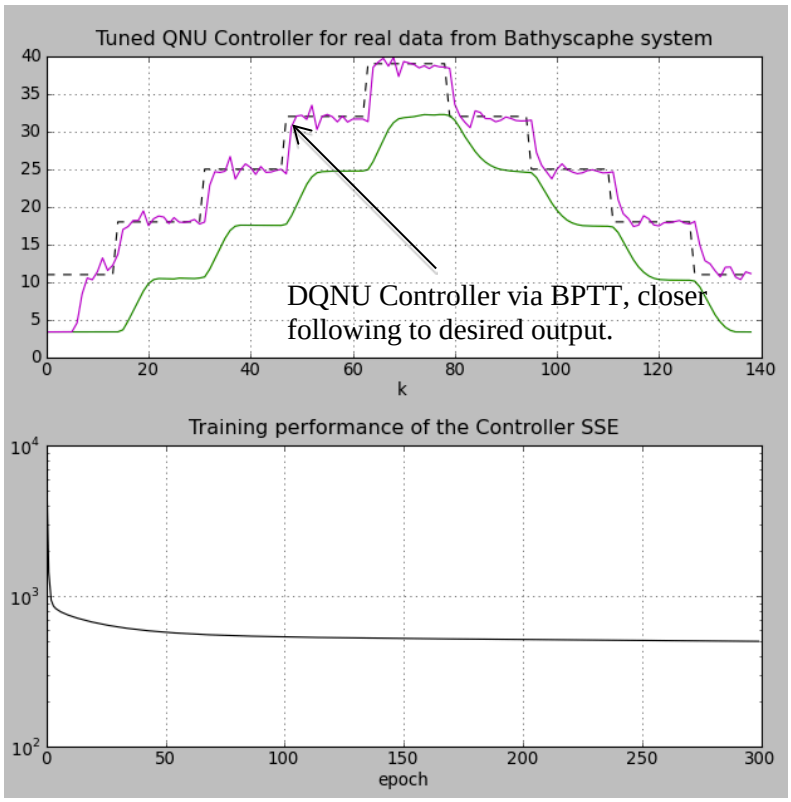

Figure 13: *DQNU_BPTT Neuro-Controller Using DQNU_BPTT Plant by designed program, there are strong potentials for control optimization of the system.*

Results of the developed software in Figure 12 and Figure 13 of the system Bathyscaphe indicate large potential for control optimization of its control loop, and indeed the Bathyscaphe system had been (with the same real data and gradient-descent based principle) optimized in [10] (Figure 1 above) and QNU controller was superior to PID and working in the full range of output values.

4. Conclusions

As analysed in the results section of this paper we can see a direct reflection of several theories presented in section 2. As seen in the plant identifications, with proper tuning of the key parameters of the DQNU, we find that it can achieve faster adaptation and consequently obtain a more precise model over the same epochs as the DLNU adaptation. Another phenomenon witnessed was the smooth behaviour of the BPTT training method for noisy data, which as mentioned in section 2, is due to the main governing law of this method concerning the input and output of the neural unit over the epochs rather than over sample-by-sample.

The essence of this software is to investigate adaptive control potentials for the process data; it was found that all though the DLNU_RTRL Neuro-Controller with DLNU RTRL training, provided an acceptable result, the DQNU with RTRL training, followed by a Neuro-controller via. DQNU with RTRL, seemed to work best, as shown above. However as mentioned previously in this paper, this result like all produced via this software is a tool for seeing the potentials of control via gradient descent based algorithms, and thus should be checked with the real system, whether such control is really possible for real application on the investigated engineering process.

List of Used Symbols

μ- Learning Rate,
ξ- Input vector into Neuro-Controller
ε- Normalisation Constant
r_o -Estimated gain of P-controller
η - Normalised Learning Rate

References

[1] Gupta, M.M., Liang, J. and Homma, N. [2003], "*Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*," IEEE Press and Wiley-Interscience, published by John Wiley & Sons, Inc.

[2] Bukovsky, I., Bila, J., Gupta, M., M, Hou, Z-G., Homma, N.: "Foundation and Classification of Nonconventional Neural Units and Paradigm of Nonsynaptic Neural Interaction" in *Discoveries and Breakthroughs in Cognitive Informatics and Natural Intelligence* series *Advances in Cognitive Informatics and Natural Intelligence (ACINI)*, ed. Y. Wang, IGI Publishing, Hershey PA, USA, Nov.. 2010. ISBN: 978-1-60566-902-1, pp.508-523

[3] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[4] P. J.Werbos, "Backpropagation through time: What it is and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[5] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[6] Bukovsky, I., Ichiji, K., Homma, N., Yoshizawa, M.: "Testing Potentials of Dynamic Quadratic Neural Unit for Prediction of Lung Motion during Respiration for

Tracking Radiation Therapy", *WCCI 2010, IEEE Int. Joint. Conf. on Neural Networks IJCNN*, Barcelona, Spain, 2010.

[7] Bukovsky, I., Rodriguez, R., Bila, J., Homma, N.: "Prospects of Gradient Methods for Nonlinear Control", *Strojárstvo Extra*, MEDIA/ST, s.r.o. publishing house, 2012, ISSN 1335-2938.

[8] Gupta, M., M., Bukovsky, I., Homma, N. , Solo M. G. A., Hou Z.-G.: "Fundamentals of Higher Order Neural Networks for Modeling and Simulation", in *Artificial Higher Order Neural Networks for Modeling and Simulation*, ed. M. Zhang, IGI Global, 2012.

[9] Bukovsky I., S. Redlapalli and M. M. Gupta : *Quadratic and Cubic Neural Units for Identification and Fast State Feedback Control of Unknown Non-Linear Dynamic Systems*, Fourth International Symposium on Uncertainty Modeling and Analysis ISUMA 2003, IEEE Computer Society, 2003, Maryland USA, ISBN 0-7695-1997-0, p.p.330-334

[10] Rodriguez , R., Bukovsky, I., Homma, N.: "Potentials of Quadratic Neural Unit for Applications", in *International Journal of Software Science and Computational Intelligence* (*IJSSCI*) ,vol 3, issue 3, IGI Global, Publishing, Hershey PA, USA ISSN

[11] Ladislav Smetana: *Nonlinear Neuro-Controller for Automatic Control Laboratory System*, Master's Thesis (sup. Ivo Bukovsky), Czech Tech. Univ. in Prague, 2008