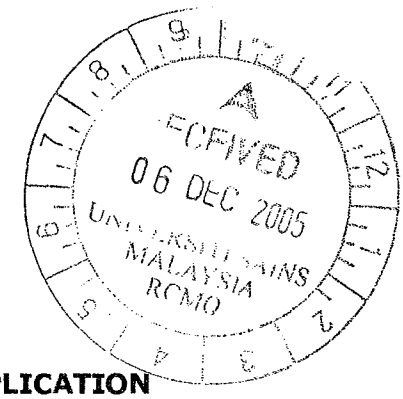


MANUAL

RECURRENT NEURAL NETWORK APPLICATION FOR ON-LINE REAL-TIME CONTROL OF NON-LINEAR DYNAMIC SYSTEMS

FUNDED BY

USM FUNDAMENTAL RESEARCH GRANT SCHEME (FRGS)

ABDUL HAMID ADOM

AHMAD NAZRI ALI

MOHD YUSOFF MASHOR

NORASHID AZIZ

FOREWORD

This manual is intended as a guide to researchers wanting to implement a type of partially recurrent neural networks called Local-Output feedback Locally Recurrent Globally Feedforward (LOLRGF) for the on-line real-time control of non-linear dynamical systems. The work is based on an Internal-Model Control (IMC) structure implementing recurrent neural networks indirectly as the controller. The configuration of the codes given is for the control of SISO systems. The work featured in this manual was part of the results from USM Fundamental Research Grant Scheme (USM FRGS).

Users of this manual will need to go through the procedures to utilise the control algorithm. This manual also assumes some basic knowledge on the part of the reader in control theory and artificial neural networks.

NOTE: The work under the USM FRGS was only completed up to the simulation part only. Other considerations when implementing the control algorithm on real-processes are discussed in the further work section in Chapter 10.

TABLE OF CONTENTS

1. INTRODUCTION

In the field of systems control, there have been many attempts to implement alternative and novel approaches to replace conventional methods. The introduction of the new methodologies and algorithms are explored in the pursuit for better control performances in as many applications as possible covering as many types of systems as possible. One of the areas that are of interest in this research is the control of non-linear dynamical systems on-line in real-time using artificial neural networks.

This manual presents the procedures, along with the necessary considerations, of implementing a type of recurrent neural networks for the adaptive control of non-linear dynamical systems. The control algorithm described in this manual is for on-line real-time type implementation.

The manual is presented in 10 chapters. Chapter 1 provides the overview of neural networks and past implementations in control with the objectives of the work conducted discussed in Chapter 2. A brief discussion in control systems and neural networks follow in Chapters 3 and 4 respectively. Neural network applications in control and on-line real time control discussions follow in chapters 5 and 6.

Detailed procedure of how to implement the work conducted in this research follows in Chapter 7. This chapter discusses all the considerations needed to implement the control system. In Chapter 8 presents a sample implementation of the algorithm on a Continuously-Stirred Tank Reactor (CSTR) process. Chapter 9 the manual discusses the research conducted as well as the implementations of the proposed algorithm. Chapter 10 poses new challenges as a continuation of the work already conducted. Finally, the Appendix lists the programmes written in MATLAB for the use in this research and the parameters for the CSTR process. Softcopy of the codes can be obtained by contacting the author at *abdhamid@kukum.edu.my*.

2. OBJECTIVES

The work conducted in this project was actually a continuation of a Ph.D. thesis on the use of neural networks for the control of non-linear dynamical systems. There were, however, some limitations with the work conducted and this project was undertaken to :

i) propose an algorithm to enable the use of neural networks in the on-line real-time control of non-linear dynamical systems.
ii) produce a manual of the implementation of the work conducted

3. CONTROL SYSTEMS

The ultimate aim of a control system is to provide the appropriate inputs to enable a plant or process responds appropriately with the given reference and to comply to some given specifications. Simple systems may require simple control systems, and a control system should not be more complicated than needed. In other words, the simplest control system that satisfies all the design and performance specifications should be adopted.

More complex systems may require more complex controllers or control methodologies. Non-linear dynamical systems may pose some problem to conventional control approaches. In conventional control methods, say PI or PID, the control of non-linear systems can be implemented by using linearised models of the system, and using different controller parameters for different operating points or ranges. More advanced control methods, for example using neural networks, may fare better. This is because, neural networks are able to model the dynamics of a plant or process, and hence there is no need for several controllers for the system to operate freely within its operating range. Only one controller is needed for the process or plant to operate at different operating points.

The above is only true if the dynamics of the plant or process stays unchanged over time. However, system dynamics will, inevitably, deviate with time due to many reasons; valves get corroded, build-up in pipes, etc. This will change the dynamics of the plant or process itself, and then even the neural network models will no longer be valid.

Previous works using neural networks will require the neural model of the process being re-trained, and usually off-line, so that an update on the weights can be performed. This is where the problem lies: updating of the network is not automatic and time-off operation is needed in order to update the neural model.

Due to this reasons, the project aims to propose a new approach to use neural networks to allow the controller to update itself in response to the changes in the

process dynamics on-line in real-time.

4. NEURAL NETWORKS

One of the favourable characteristics of neural networks is its ability to generalise and represent the dynamics of a process once the model has been trained using the input-output data of the said process. The approach in which this model is used in the control system is varied, and may be used directly and indirectly in the control loop.

The method of using neural networks in the control loop in place of the more conventional P, PI or PID controllers provides a more "universal" controller in that the controller produced using the latter covers the possible operating range of the process. For the P, PI or PID controllers, gain scheduling is needed to handle the different operating points.

This advantage is due to the neural networks' characteristics of generalising data. However, the model is only as accurate and reliable as the data used to train it. Another favourable characteristic of neural networks is its ability to optimise cost functions. This very characteristic form the fundamentals of the work conducted in this research.

When using neural networks model of the process as the controller, the performance of the control systems are bounded by the accuracy of the neural network model(s) used in the control loop. The accuracy of the neural network models are dependent upon the accuracy of the data used to represent the system. In the case of using the inverse model of the system as the controller, the error can be significant. This is due to the fact that the inverse model is obtained from the forward model.

The process of obtaining the forward model will result in some modelling error, albeit a very small one. However, the inverse modelling of a process is more difficult, and almost always will result in a much greater error compared to the forward modelling. This means that in the process of obtaining the inverse model through the forward model will result in a rather significant error.

In addition to the above arguments, the neural network considered in this work is of the recurrent type. Basic feedforward neural networks, like the Multi-Layer Perceptron (MLP), are good for use with static problems, say pattern recognition for example. It

can, however, represent non-linear dynamical systems by means of external tapped-delay lines (TDL). These TDLs will incorporate memory into the network to enable it to represent the dynamic properties of the process or plant.

Recurrent neural networks on the other hand incorporate internal memory lines within the structure, and hence the dynamical information about the process or plant is inherent. This usually results in better modelling accuracy and generalisation. Fully Recurrent Neural Networks (FRNN) are too complicated, and usually most of the feedback connections are redundant. Partially recurrent neural networks (PRNN) offers a better compromise between the favourable characteristics of recurrent networks and simpler network structure.

In the past, we have implemented a type of recurrent neural networks, namely the Local-Output feedback Locally Recurrent Globally Feedforward (LOLRGF) neural networks in the IMC control strategy. This particular structure was based on the Frasconi-Gori-Soda architecture (Tsoi and Back, 1994; Campolucci *et al.*, 1999) and was introduced among other reasons to simplify the learning compared to a Fully Recurrent Neural Networks (FRNN) (Campolucci *et al.*, 1999).

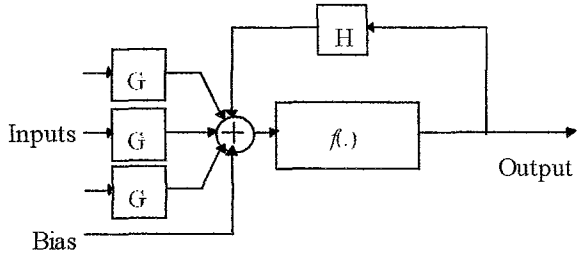


Figure 1. Local Output feedback Locally Recurrent Globally Feedforward (LOLRGF) neural network hidden layer neuron. The feedback and the input to the hidden neuron are passed through polynomial functions, which are basically FIR filters, defined as H and G respectively. *f*(.) is the non-linear activation function. In the MATLAB code given, hyperbolic tangent activation function is used.

The LOLRGF neural networks take form of the conventional Multi-Layer Perceptron (MLP). However, the nodes in the layer are what is called the Local-Output feedback neurons as shown in Figure 1.

Another advantage of the proposed control algorithm is that it does not use the neural network as the model or the inverse model of the process. It is used as the optimiser

to minimise the cost function generated by the difference between the process output and the reference. Hence there is no constraint on the accuracy of the neural network models. This approach also has the additional advantage in that the tedious process of plant modelling, although not totally eliminated, is simplified.

5. NEURAL NETWORKS IN CONTROL SYSTEMS

There have been many applications as well as attempts to use neural networks as part of the control strategy. Recurrent neural networks has the apparent advantage over linear control approaches in that the models used in the control loop represents the plant or process over its dynamic range. That means, even if the operating range changes, the model is still valid and should provide satisfactory control actions.

However, if the dynamics of the process itself changes, due to many reasons such as rusty pipes etc., the model is no longer valid. This means that the model will have to be re-trained. The re-training is needed to be conducted off-line, and will require as much data as the training of the previous model. This data is not always available. Also, the changes in the process dynamics are often too small to excite the learning in neural networks although it sometimes results in significant change in control action performance.

The work conducted in this research performs the updating the neural networks incrementally as the process is running. Theoretically, if there is any change in the process dynamics, there will also be difference between the process output and the reference. This will result in the neural network updating itself appropriately.

6.0 ON-LINE REAL-TIME CONTROL

On-line real-time control needs the controller to have the ability to adapt to the changes in the process being controlled. Real process may experience changes in dynamics due to wear in components etc. This will result in change in the rate of flow, speed of rotation and so on. All these factors will contribute to the changes in the process dynamics.

Neural networks are able to handle changes in process operating range well, but the changes in the process dynamics will affect the controller performance. The main

objective in this research is to find a method to enable the neural networks used in the control loop to be able to adapt itself fast enough as the process is running so that it is able to adapt itself on-line in real-time. The proposed system uses the Recurrent Prediction Error algorithm (Adom, 2001).

One note of caution is the speed of the response of the system. So far the work conducted has only involved simulated processes. When dealing with real processes, requirements when dealing with fast systems, as in electric motors, are different to systems with slow or very slow response as in fermentation. Since the codes were implemented in MATLAB, there exist some limitations when controlling systems with very fast time constants.

7.0 ALGORITHM IMPLEMENTATION PROCEDURES

This section discusses the implementation procedures of the control algorithm for single-input single-output (SISO) systems. The procedures are presented step-by-step for easy implementation by user.

Naturally, to implement any control strategies, there needs to be considerations in terms of the process or plant configurations, controller parameters etc. Also, when dealing with neural networks, other parameters, such as network architecture, size, learning algorithm and its learning parameters are crucial.

Another important aspect is the data conditioning. Performing data conditioning avoid misclassification of the data and biased results due to large magnitude input-output data.

The section first discusses the considerations for the overall control system. Then we consider the process or plant. Next comes the considerations needed for the neural networks itself, along with the training algorithm.

7.1 CONTROL SYSTEM CONSIDERATIONS

As with all implementations using neural networks, the most important issue is the input-output of the system.

The choice of the network size/structure will be case dependent. This is because the number of inputs and outputs of the neural networks will depend on how many inputs and outputs the system has, and also all other inputs that the system needs in order to represent the system as well as additional data needed.

7.2 PROCESS INPUT-OUTPUT CONSIDERATIONS

Neural networks can learn the dynamics of a plant or process by means of their input-output pairs. This means that neural network modelling is a black-box modelling and it does not need to know the states of the plant or process. Because of this, the reliability of the input-output data pairs are crucial to ensure the success of the control scheme. We will look at the factors to be considered one-by-one.

7.2.1 PROCESS INPUT-OUTPUT PAIRS

The first consideration is the process input-output pairs themselves. The use of neural networks will require the determination of the number of inputs and outputs to the system. This will be used to determine the number of inputs and outputs to the neural network model. Also, other inputs necessary for the neural model to capture the system dynamics will also have to be considered.

Once the inputs and outputs to the process have been determined, the data collection can be conducted. The plant should be given a set of inputs that covers the whole operating range of the plant and the corresponding set of output responses of the plant are recorded. These will form the input-output data pairs of the plant.

7.2.2 DATA CONDITIONING

The neural networks used in the control system will only be as good as the data used to train or update it. Since the structure of the neural networks is basically connections with certain gains (in this case the weights), utilising certain linear/non-linear functions, users must be very careful as to not saturate the outputs or even the hidden node outputs. If this happens, the network will not be able to update itself using the algorithms. To make sure that this does not happen, the data must be conditioned first.

The data should be scaled so that it has a ***mean of zero and variance of 1***. This will not only eliminate the problem of the network outputs saturating, but will also eliminate the problem of bias due to certain data with large amplitudes.

$$x_{scaled} = \frac{x - x_{mean}}{x_{std}}$$

where x_{scaled} = scaled data
x = x vector
x_{mean} = mean of vector x
x_{std} = standard deviation of x

As a start, the user will have to collect certain number of input-output data pairs and find the scaling needed to bring the data pairs to have a zero mean and variance of 1. This scaling will then be used throughout the running of the system. This has to be conducted this way because during the running of a system on-line in real-time, the data will have to be scaled as it comes in, i.e. one at a time. Hence, it is not possible to find its variance nor mean. So the scaling of the pre-collected data will be used to scale the incoming data into the system.

7.3 CONTROL SYSTEM IMPLEMENTATION

The nature of using neural networks requires the user to determine the network parameters before implementation. The determination for the network parameters will depend on the process or plant. The inappropriate choice of learning parameters of the initial condition may result in the algorithm not converging, even though a suitable structure and training algorithms were used. The problem is there are no rules in determining the optimum learning parameters for the training algorithm.

After a number of runs of the training, these produced results may suggest a smaller range of learning parameters that produce the most desirable results. By using these smaller range of parameter values, the length of the experiment for a particular structure-algorithm pair was made shorter without having to repeat the procedure unnecessarily.

7.3.1 PARAMETER SETTINGS

Users will have to determine the suitable parameters for the neural networks as well as the training algorithm. There are several methods are available to determine the network size, for example using a pruning algorithm. This algorithm will 'prune' the

network of a large size until it reaches the smallest network size without the los of significant accuracy.

The work conducted in this research uses the more conventional trial and error method of building the network size by a pre-determined steps and testing each of the network for its accuracy.

7.3.2 NEURAL NETWORK PARAMETERS

The network parameters mainly concern the network size. But since the number of network inputs and outputs are case-dependent, and are predetermined and fixed throughout the procedure, the main parameter that can be chosen is the number of hidden nodes. The number of hidden nodes are often referred to the size of the network since the number of inputs and output are fixed.

The size of the neural network used will be problem-dependent. Some testing can be conducted to test the suitability of the number of hidden nodes, for example, to a certain application. To do this, the system can be simulated to run using the tested network. If, after a few iterations, say 100 or so, and the system behaves as required, then that network size can be used. If not, then different network size may be tested.

Usually, the test starts with a smaller network, say 4 hidden nodes, and gradually more nodes are added if the initial setting was not satisfactory. The usual numbers are 4, 6, 8, 10 and so on.

Another parameter that may be changed is the number of delayed hidden nodes output are fed back into the hidden nodes (refer to function H in Figure 1). However, this is usually unnecessary because the structure given in the sample MATLAB code was found to produce best results (Adom, 2001).

7.3.3 ALGORITHM PARAMETERS

For the implementation of on-line real-time control of systems, batch learning or updating of the network weights is not applicable. This is because the network will have to keep track of the difference between the process output and setpoint or reference at every sample time. Hence the updates of the network weights will have

to be performed at every sample. This means that only iterative weight updates can be implemented.

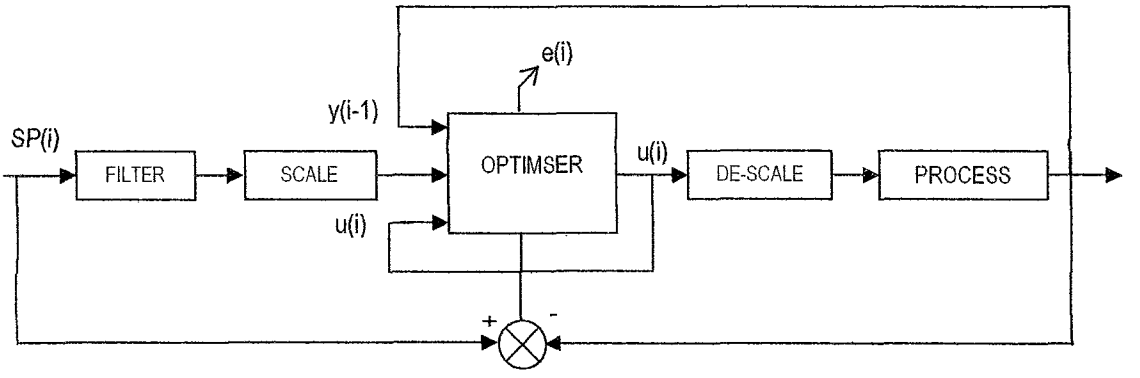


Figure 2. Control system configuration

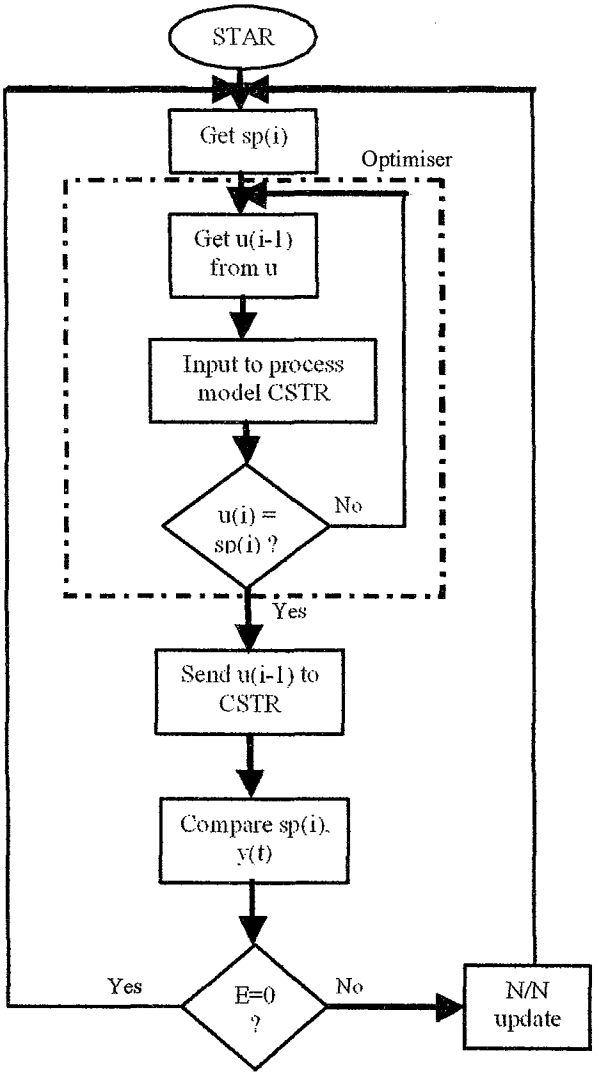


Figure 3. The flow-chart of the control algorithm implemented

7.3.4 PROGRAMME RUNNING MONITORING

Another important factor during the testing of the network in the control loop is the monitoring of the network and system performance on-line. As for the network, this can be achieved by monitoring its sum-of-squared-error (SSE).

In conventional approach of using neural networks where they are pre-trained off-line before being integrated in the control loop, the SSE of the network validation is used and not the SSE of the network during training. This is because, during the network training, the algorithm pushes the network output to follow the reference, and usually do not show the true picture of the state of the network training (Adom, 2001). The SSE during network validation provides a true picture since during the validation process, the network weights are kept fixed. Think of it as the results of students tested during an examination. What he or she answers during the examination is what he or she knows and have learnt.

Another parameter that can be used to monitor the performance of the network is the hidden node output(s). The hidden node output should not saturate during the network training, or in this case, during the running of the control process. If this happens, no updates of the weight can be achieved.

7.4 SUMMARY

The implementation of the algorithm will require the considerations of the process input-output pairs (data conditioning), control system and neural network parameters (which include the algorithm parameters) as well as programme running monitoring.

The system parameters, as in the number of plant inputs and outputs, are pre-determined and fixed throughout. Other parameters, such as the delayed plant inputs and outputs, the difference between the plant output and reference, can be used as part of the controller input.

The parameters of the system, as in the determination of the input and output are predetermined. This will determine the number of inputs to the neural networks.

8.0 SAMPLE APPLICATION

This section presents a sample implementation of the proposed algorithm. The test-bed used is a Continuously-Stirred Tank Reactor (CSTR) process.

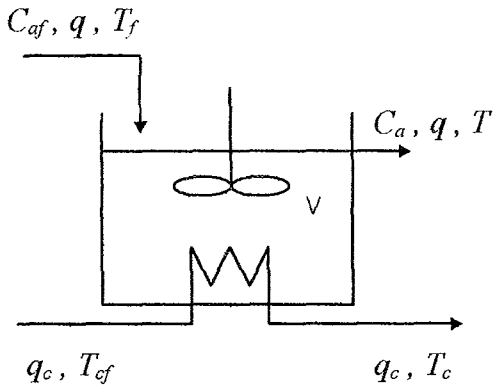


Figure 4. Schematic diagram of a Continuous-Stirred Tank Reactor (CSTR process). The input and output of the tank are q_c and C_a respectively .

The process in a single-input single-output (SISO) system of highly non-linear relationship between the input and output. This process was chosen because it provides a good challenge for the neural network models both in modelling and control due to its non-linearity. It has highly non-linear dynamical properties, which are described by two non-linear differential equations:

$$\frac{dC_a}{dt} = \left(\frac{q}{V}\right)\left(C_{af} - C_a\right) - k_0 C_a \, exp\left(\frac{-E}{RT}\right) \quad (1)$$

$$\frac{dT}{dt} = \left(\frac{q}{V}\right)\left(T_f - T\right) + \left(\frac{-\Delta H k_0 C_a}{\rho C_p}\right) exp\left(\frac{-E}{RT}\right) + \left(\frac{\rho_c C_{pc}}{\rho C_p V}\right) q_c \left[1 - exp\left(\frac{-hA}{q_c \rho_c C_{pc}}\right)\right]\left(T_c - T\right) \quad (2)$$

where the process inputs and outputs are the coolant flow rate, q_c, and the effluent concentration of species A, C_a, respectively. Refer to Appendix 2 for the descriptions of the other system parameters.

The initial conditions of the CSTR process are assumed to be a nominal steady-state where the concentration and tank temperature have the following values :

C_a = 8.235x10^{-2} mol/l , T = 441.81K

8.2.2.2 Filter constant, *f*

A range of setpoint values was given as the input to the IMC scheme. Referring to figure 1, the filter constant was varied to study its effects on the control of the IMC scheme. The filter constant (*f*) can be selected to reduce the gain of the feedback system moving from the perfect controller. The choice of filter is an important in implementing the IMC scheme to ensure proper control action is achieved. This choice will give an affect on the transient response of the process. The filter will keep the input signal to the controller bounded. The filter used in this experiment is a discreet-time first exponential filter.

The correct choice of filter constant will help dump the response of the process to produce a smooth transition to follow the change in the setpoint. This done by tuning the filter constant using trial and error method.

8.2.2.3 Forgetting factor, *ff*

The forgetting factor used in the training algorithm (RPE) for this experiment was kept constant at unity. By doing this its allow the algorithm to fully converge and observed to give satisfactory performance. The value range used in this experiment is 0.1 to 1.

8.2.2.4 Optimiser output change constant, *c*

This approach substitutes the inverse model with an optimiser as the controller in the control loop. The optimiser will find a suitable output i.e. input to the process such that its gain is the exact inverse of that of the forward model.

As an optimiser was used as a controller in this IMC scheme, the optimiser output change constant will put into consideration if using the extended cost function. In this experiment extended cost function was used.

8.2.2.5 Length of training, *np*

The training length will give an effect on the learning process, which the input data will be repeated.

8.2.2.6 Setpoint, *sp*

There are two sets of input data (setpoint) i.e. randomly generated and repeatition data. In this experiment repetition setpoint was used to detect the network learning during the training process.

8.2.3 Network connections

There are three networks connection used in this experiment i.e. setpoint, process output and optimiser output. All these three will be used as an input to the controller (optimiser).

The IMC scheme was tested separately before being integrated into the control loop. This was to ensure that each sub-system representing each block produced the appropriate response to corresponding input signals. In this way, troubleshooting the IMC schemes are made easier and any faults results in the sub-systems can be properly identified. The overall implementation of the control schemes was done in MATLAB. On the other hand, the simulated CSTR process was implemented in SIMULINK. To communicate between the process and the IMC scheme *.m* file was implemented and MATLAB integration function *rk45* (Runge-Kutta order 5) was used.

After the completion of the network and training parameter selection was completed, the testing of the performance of the selected controller was conducted. The results were then compared to Hybrid MultiLayer Perceptron (HMLP) conducted using the same procedures. The extra work was conducted to validate the superior perfromnace of the LOLRGF neural networks.

8.3 Results and Discussions

8.3.1 Network size

Table 1: Result for Network size according to time for 500 samples

Network Size	Time (s)
8	380.4840
12	392.7060
16	405.9220
20	428.7660

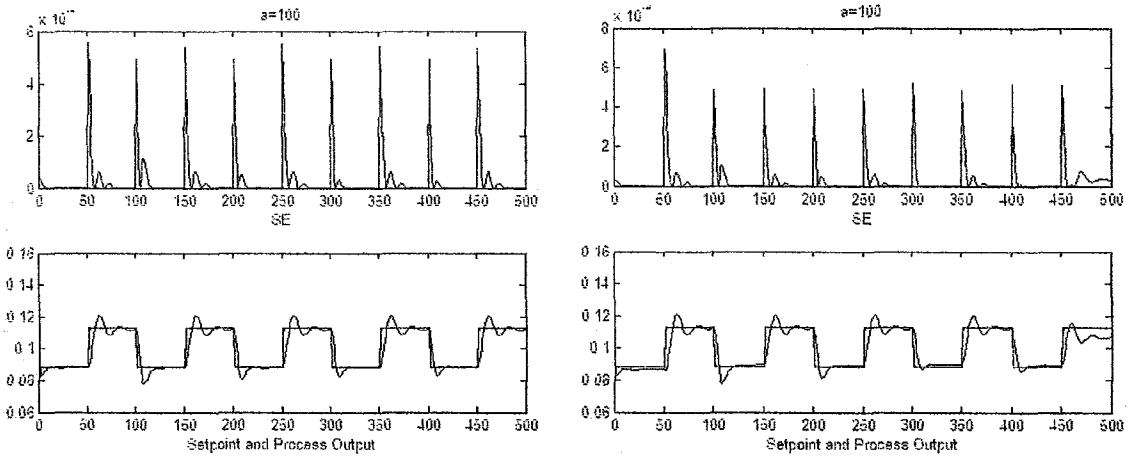
In this experiment there are 4 network sizes used i.e. 8, 12, 16 and 20. Table 1 at the above shows the result for network size according to time for 500 samples. The results

of the investigation into the effect of increasing the LOLRGF network size on the resultant model accuracy were as expected. It was observed that an increase in network size will gives an increase in model accuracy. However, the higher the model order, the larger the number of weights in the mode, hence the training time increases significantly. Differences in the number of weights in the network affects the training time significantly, especially with larger network size.

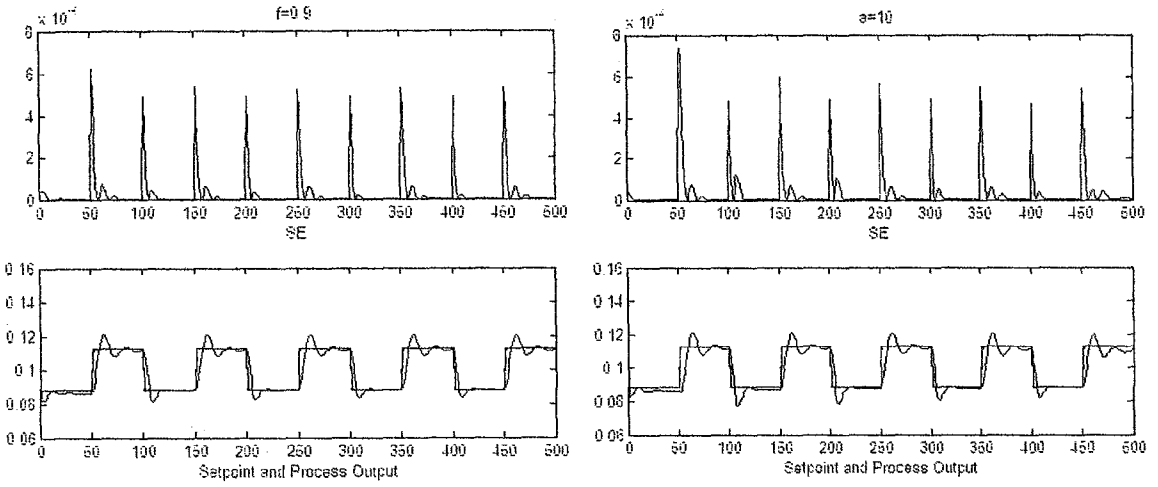
8.3.2 Learning parameters

The highest accuracy was found using a trial and error method. This process was lengthy, as the training has to be repeated each time using different learning parameters. The increase in model accuracy will cause the expense of increased network model training times, which increases significantly. The training period increased with a higher ratio than the increase in the network accuracy.

8.3.2.1 Feedback gain (a)



(a) (b)



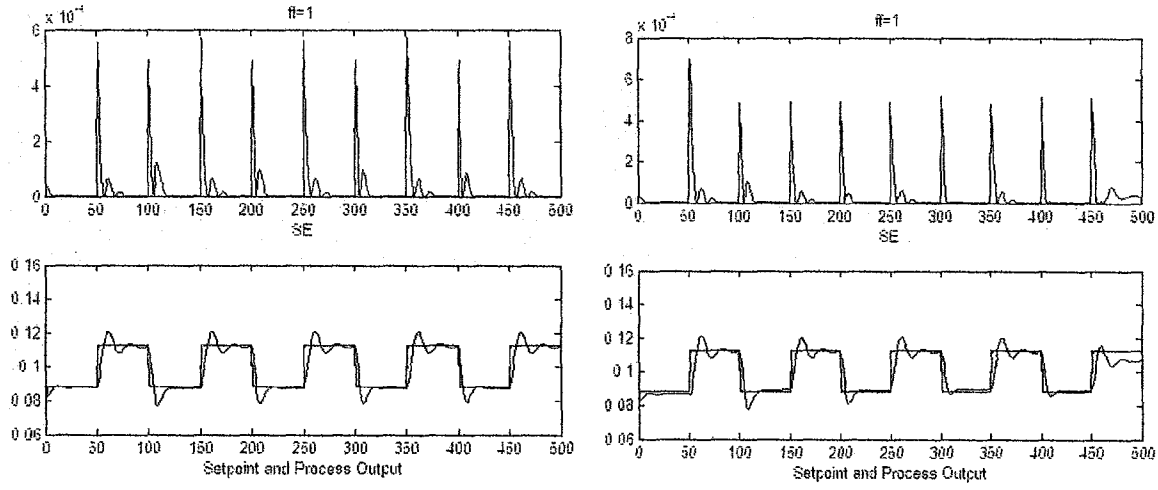
(c) (d)

Figure 7: (a) network size 8 (b) Network size 12 (c) Network size 16 (d) Network size 20

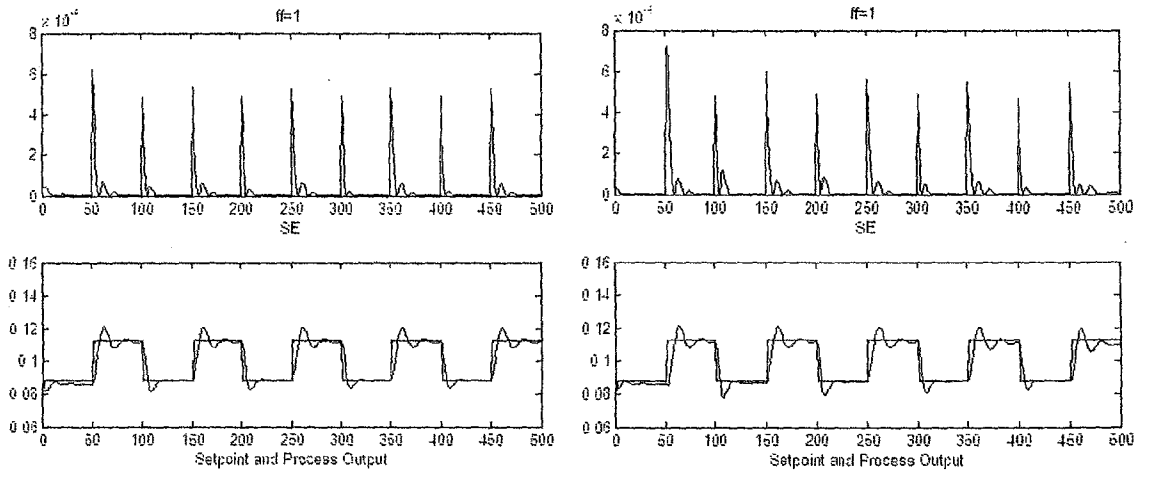
Figure 7 at the above shows the best output for error, process output and setpoint corresponds to the filter constant value. Its shows that all of the network size gave best result when filter constant value is 0.9 other than that gave the result which shows the network cannot reduce the gain of the feedback system moving from the perfect controller. As the result this filter constant value for each network size was use as a set value for next learning parameters processes.

The filter constant is very important in determining the process response. A small value results in a slow but smooth process response, and a high value results in fast process response but at the expense of an oscillatory controller output.

8.3.2.3 Forgetting factor (*ff*)



(a) (d)

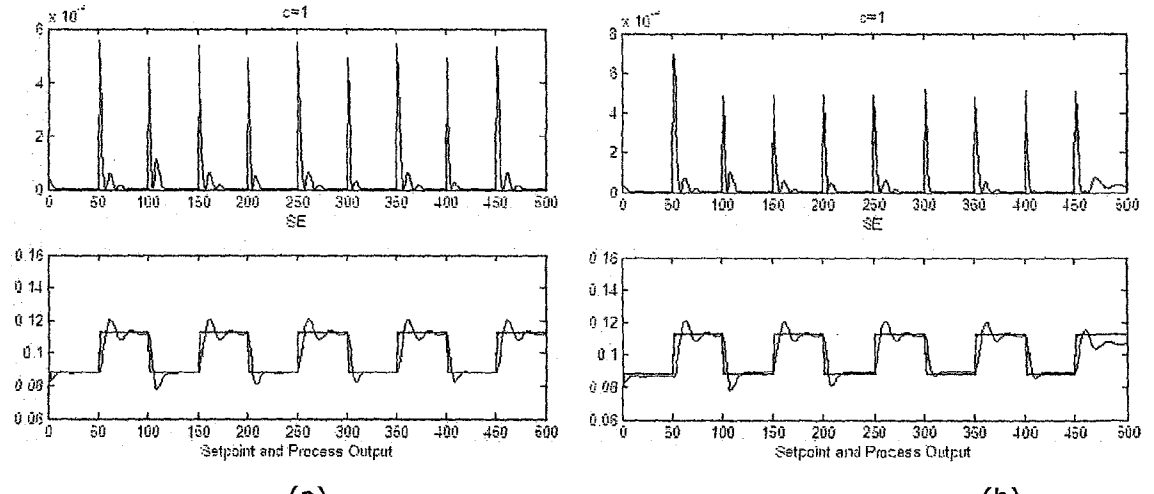


(c) (d)

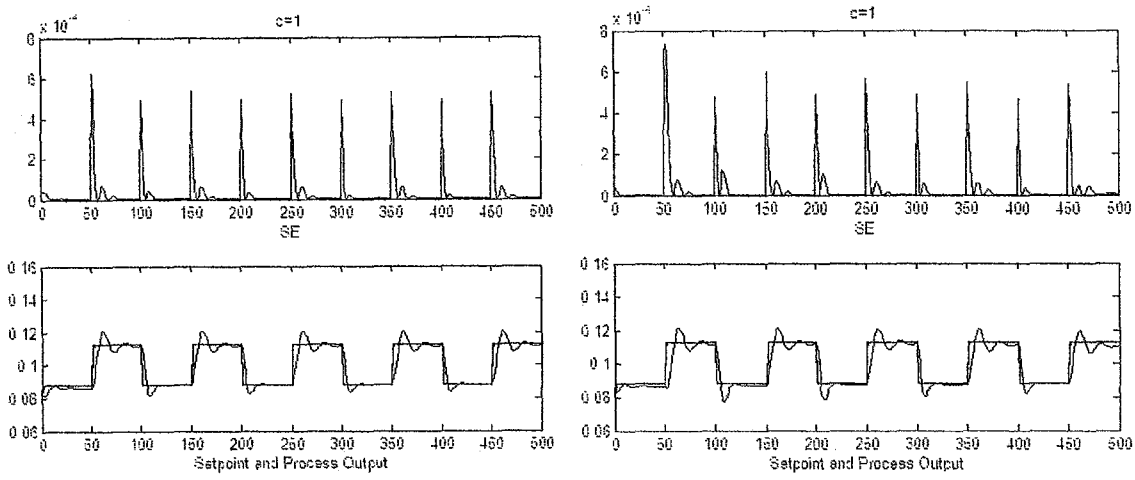
Figure 8: (a) network size 8 (b) network size 12 (c) network size 16 (d) network size 20

Figure 8 at the above shows the best output for error, process output and setpoint corresponds to the forgetting factor. It's shows that all of the network size gave best result when the forgetting factor value is 1. Due to that this value was used for each network size for next learning parameters processes.

8.3.2.4 Optimiser output change constant (c)



(a) (b)



(c) (d)

Figure 9: (a) network size 8 (b) network size 12 (c) network size 16 (d) network size 20

The effect of the past optimiser output on the current output can be controlled by varying the optimiser output change constant, c.

Figure 9 at the above shows the best output for error, process output and setpoint corresponds to the optimiser output change constant (*c*). It's shows that all of the network size gave best result when the *c* value is 1. Due to that this value was used for each network size for next learning parameters processes.

8.3.2.5 Length of training (*np*)

Table 2: Length of training for network size 8

Length of training	Time (s)
5	371.4220
10	380.4840
20	384.4370
50	412.9690

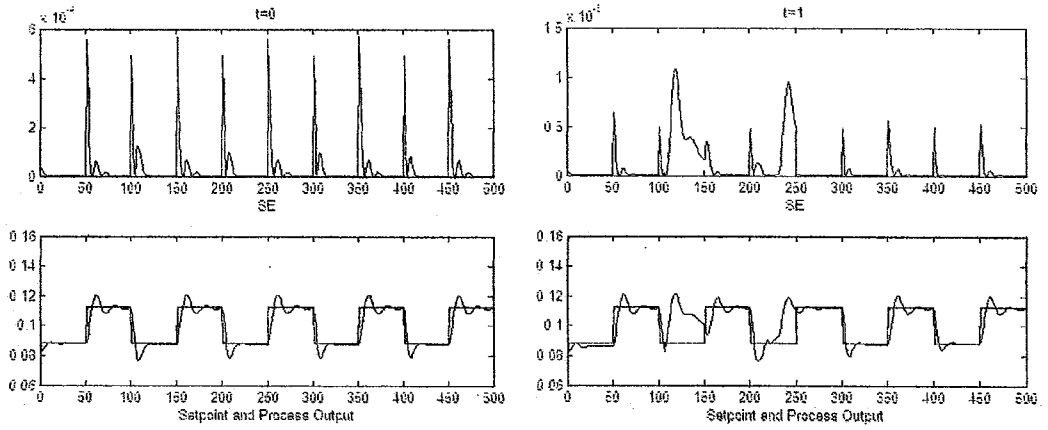
In the learning parameter it will give an effect on the learning process, which the input data will be repeated according to the length of the training. By increasing the length of training it will reduce the error however higher value used will give an affect on the learning process which will take longer to complete. Table 2 at the above shows the times which take for every length of training used. In this experiment the length of training used is 10 seems it will take approximately as 0.01m/s in the realtime.

8.3.2.6 Setpoint (*sp*)

In this experiment repeatation setpoint data was used to detect the network learning during the training process. It seems with suitable learning parameters used in the experiment, gave better result and reduce the error. For this experiment number of input data (setpoint) used is 500 data. By increasing the number of data it will show better result and could trace how long it take to reduce the error to minumum condition.

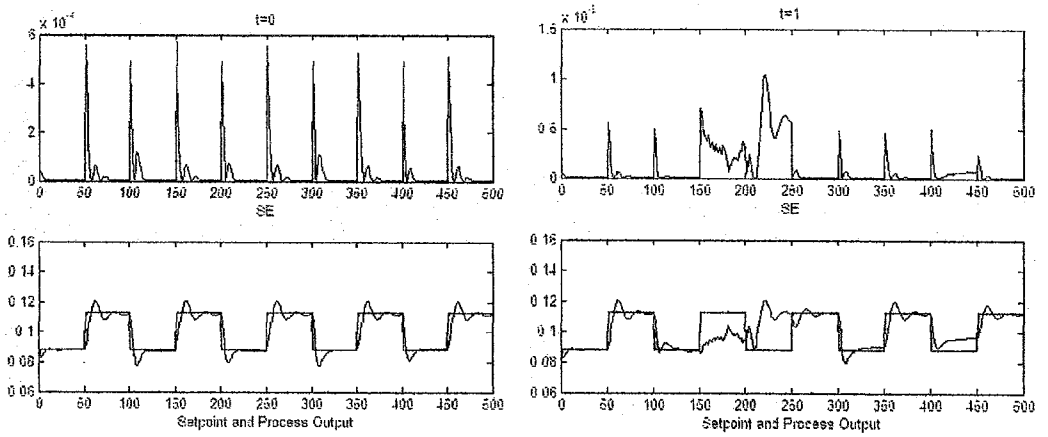
8.3.3 Network connection

There are three networks connection which used in this experiment i.e. setpoint, process output and optimiser output.



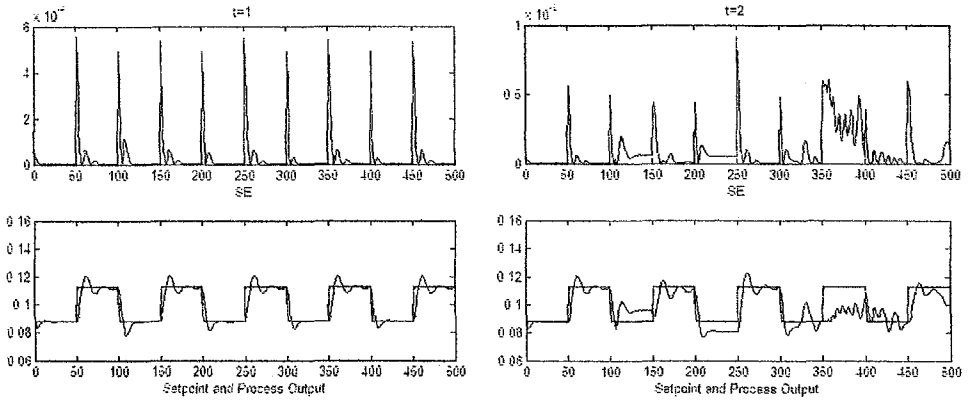
(a) (b)

Figure 10: (a) NN inputs sp(t), y(t-1), u(t). (b) NN inputs sp(t), sp(t-1), y(t-1), u(t).



(a) (b)

Figure 11: (a) NN inputs sp(t), y(t-1), u(t). (b) NN inputs sp(t), y(t-1), u(t), u(t-1).



(a) (b)

Figure 12: (a) NN inputs sp(t), y(t-1), u(t). (b) NN inputs sp(t), y(t-1), y(t-1), u(t).

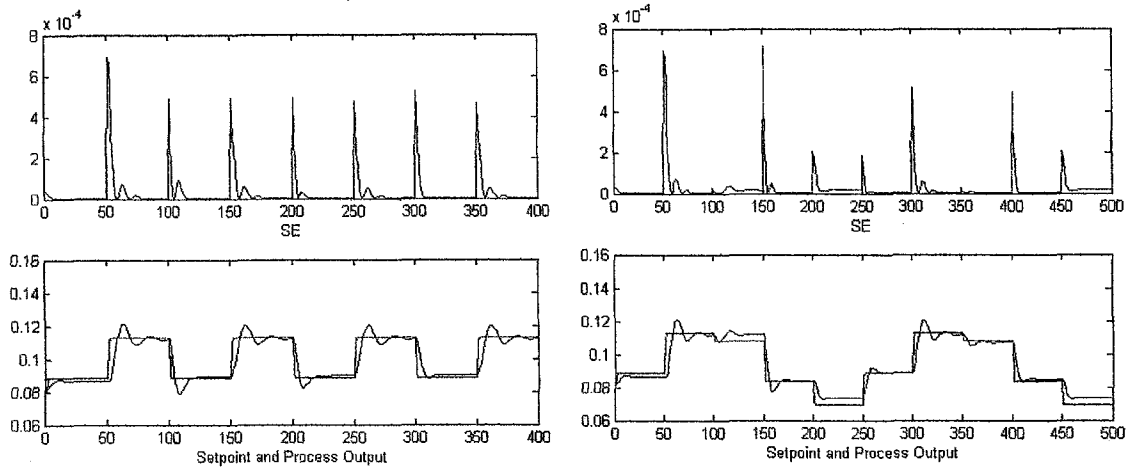
Figures 10 - 12 above show the output for setpoint, optimiser output and process output at t. For Optimiser output and setpoint was started at t=0, while for process output delayed at t=1. Using this setting the output result will show less errors compared if start at t=1 or more.

From the results, the inputs to the network that provides the most suitable control actions is sp(t), y(t-1) and u(t).

8.3.4 Overall Results

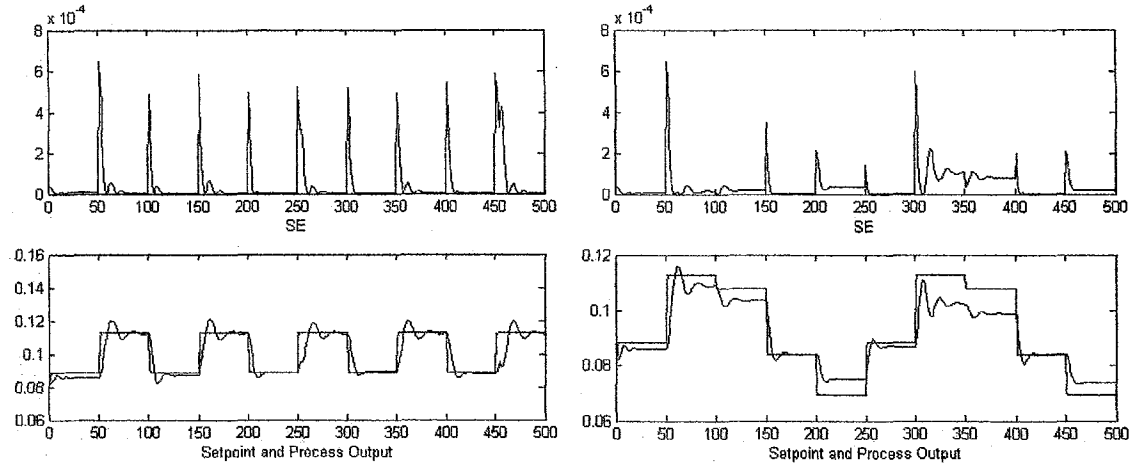
The above results represent the progression of the experiments to obtain the final neural network controller. This section presents the results if the selected controller using the LOLRGF compared to another using HMLP neural networks.

Figure 13 shows the process setpoint-tracking test when recurrent neural network, LOLRGF, was used in the controller. Figure 14 shows the same test when feedforward neural network, HMLP, was used in the controller. Both of the networks were initialised to a random set of weights without off-line pre-training.



(a) (b)

Figure 13. Setpoint-tracking test using LOLRGF neural networks in the controller.



(a) (b)

Figure 14. Setpoint-tracking test using HMLP neural networks in the controller.

Referring to Figures 13 and 14, it can be seen that the feedforward neural networks used, HMLP, is unable to cope with the non-linearity of the process dynamics. Although trained using the same RPE algorithm, the network is unable to produce satisfactory control actions. In Figure 14(b), the setpoint-tracking error seems to be greater with time as opposed to the one seen in Figure 13(b).

As can be seen in Figure 13(a), the error slowly decreases as the controller is presented with the same setpoint values. The squared error (SE) plot show the peaks

gets smaller with time. Further tests are being conducted to try to eliminate the offset still visible as seen in Figure 13(b).

Tests are being done to control a real process, in this case a DC motor, directly from MATLAB. This is achieved via BASIC Stamp module being developed. The test on a real process will have to be performed to verify the feasibility of implementing the proposed controller approach in real life.

9.0 DISCUSSIONS

One of the training algorithm parameter investigated during this period of the project is the length of training sample. At every sample, if the process output differs from that of the setpoint, the neural network is trained, i.e. the weights of the network are updated. The longer this training period, the better the network is supposed to be at predicting the process response. However, the longer training means that the controller will take a longer time to provide the next control signal. Hence, a compromise between the length of network training and accuracy of the control actions has to be considered. In the case of this project, the training length of the neural network was set to 10 samples. Due to the nature of implementation in MATLAB/Simulink which has a limitation on the speed of the response, the designed controller will have to be coded in C for the control of high speed applications. For process controls, the codes programmed in MATLAB will be adequate.

10.0 FURTHER WORK

The work discussed was tested on simulated process. The implementation using real processes will require other consideration and are discussed below.

When dealing with a real process, the time constant will be one of the determining factors whether the proposed control algorithm is feasible. The codes for the controller were implemented in MATLAB. This is fine for systems with slow time constant, i.e. greater than 1second.

For systems with very small time constants, like an electric motor, different approach to the implementation of the algorithm is needed. The user may choose to convert the codes into C. In this case, the user will also have to convert the built-in MATLAB

function, *fmin*. This is the function used to minimise the error between the controller and the reference.

Another consideration is if the MATLAB software is not available. This will also be the case when the whole controller system is an independent or embedded control system. Then the codes to the algorithm will have to be converted to the appropriate software language.

11.0 SUMMARY

The manual presents the findings of the fundamental research under the usm frgs grant for recurrent neural network application For On-line real-time control Of Non-linear dynamic systems

The description takes the user from the start to finish including the considerations of each case.

REFERENCES

Adom, A. H., "Modelling and Control of Non-linear Dynamic Processes using Partially Recurrent Neural Networks", Ph.D. thesis, Liverpool John Moores University, 2001.

Campolucci P., Uncini A., Piazza F. and Rao B. D., "On-Line Learning Algorithms for Locally Recurrent Neural Networks", IEEE Transactions on Neural Networks, vol. 10, no. 2, pp. 253-271, March 1999.

Tsoi A. C., and Back A. D., "Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures", IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 229-239, 1994.

Tetsuro Yabuta and Takayuki Yamada, "Neural Network Controller Characteristics with Regard to Adaptive Control", IEEE Transactions on Systems, Man and Cybernetics, Vol 22,No. 1 January/February 1992, pp 170–177

Karl Johan Astrom, "Adaptive Feedback Control", Proceedings of the IEEE, Vol 75, No. 2 February 1987. pp 185 – 217

Allon Guez, James L. Eilbert and Moshe Kam, " Neural Network Architecture for Control", IEEE Control Systems Magazine April 1988 , pp 22 - 25

K.Khorasani, " A Robust Adaptive Control Design for a Class of Dynamical Systems Using Corrected Models", IEEE Transactions on Automatic Control, Vol 39, No. 8 August 1994, pp 1726 - 1732

Kumpati S. Narendra and Jeyendran Balakrishnan, "Adaptive Control Using Multiple Models", IEEE Transactions on Automatic Control, Vol 42, No. 2, February 1997, pp 171 – 187

Gang Feng, " A Robust Approach to Adaptive Control Algorithms", IEEE Transactions on Automatic Control,Vol 39, No. 8 August 1994, pp 1738 – 1742

Jin Young Choi and Jay A. Farrell, " Nonlinear Adaptive Control Using Networks of Piecewise Linear Approximators", IEEE Transactions on Neural Networks, Vol 11, No 2, March 2000, pp 390 – 401

Kun Yuan Huang, Hong Chan Chin and Yann Chang Huang, " A Model Reference Adaptive Control Strategy for Interruptible Load Management", IEEE Transactions on Power Systems, Vol 19, No. 1, February 2004, pp 683 – 689

Xiang Jie Liu, Felipe Lara-Rosano and C.W.Chan, "Model Reference Adaptive Control Based on Neurofuzzy Networks", IEEE Transactions on System, Man and Cybernetics – Part C: Application and Reviews, Vol 34, No 3, June 2004, pp 302 – 309

Mitsuru Kanamori and Masayoshi Tomizuka, " Model Reference Adaptive Control of Linear Systems with Input Saturation", Proceedings of the 2004 IEEE International Conference on Control Applications, Taipei, Taiwan, September 2-4, 2004. pp 1318 – 1323

S.S.Ge, Fan Hong and Tong Heng Lee, "Robust Adaptive Control of Nonlinear Systems with Unknown Time Delays", Proceedings of the 2004 IEEE International Symposium on Intelligent Control, Taipei, Taiwan,September 2-4, 2004. pp 1 – 6

Wen Shyong Yu, "Model Reference Fuzzy Adaptive Control for Uncertain Dynamical Systems with Time Delays", IEEE International Conference on Systems, Man and Cybernetics, pp 5246 – 5251

Tomohisa Hayakawa, Wassim M. Haddad, Naira Hovakimyan and VijaySekhar Chelaboina, " Neural Network Adaptive Control for Nonlinear Nonnegative Dynamical Systems", IEEE Transactions on Neural Networks, Vol 16, No.2 March 2005, pp 399 – 413

Baris Fidan, Youping Zhang and Petros A. Ioannou, "Adaptive Control of a Class of Slowly Time Varying Systems with Modeling Uncertainties", IEEE Transactions on Automatic Control, Vol 50, No. 6 June 2005, pp 915 – 920

N.C.Quang, M.J.Tordon and J.Katupitiya, " A New Approach to Switching Robust Adaptive Control", 43rd IEEE Conference on Decision and Control, December 14-17 2004, Bahamas, pp 3247 – 3252

Kemal Ciliz and Ahmet Cezayirli, " Combined Direct and Indirect Adaptive Control of Robot Manipulators Using Multiple Models", Proceedings of the 2004 IEEE Conference on Robotics, Automation and Mechatronics, Singapore, 1-3 December 2004, pp 525 – 529

APPENDIX 1

MATLAB CODE

Note: The MATLAB code attached uses the built-in function *fmin* to optimise the cost function. This function can be substituted with any other optimising functions.

```
clc;
clf;
clear all;
%-------------------------------------------------------------------------------
% IMC implementation using LOLRGF neural network model and a predictor as the
controller.
% Uses MATLAB function fmin to find the best fit (optimiser output) so that the
% output of the model is closest to the filter output.
% All values used in the structure is scaled using the DCSALE function.
%
% This system is with filter with scaling
%-------------------------------------------------------------------------------
% Loading Data
**************************************************************************
******
load setpoint2;
load w20;
sp=sp(1:500);
np=length(sp);

%**************************************************************************
********

% Initialising variables

**********************************************************************
% Initialising filter variables

nh=20;                    % Hidden nodes
f=0.9;                    % Filter constant
fout=0;                   % Present output of filter
```

```
fout1=0;                % Previous value of the filter output
fout_s=0;               % Previous value of the filter output
fout_s1=0;              % Previous (t-1) scaled value of the filter output
fout_s2=0;              % Previous (t-2) scaled value of the filter output

% End initialising filter variables

% Initialising process variables

x=[0.08235 441.81];     % Initial condition of process
err1=zeros(1,np);           % Initial condition of errors
ynn=zeros(1,np);

xfwd1=rand(nh,1);       % previous hidden layer output (xh(t-1))
xfwd2=rand(nh,1);       % previous hidden layer output (xh(t-2))
xfwd3=rand(nh,1);       % previous hidden layer output (xh(t-3))


% End initialising process variables

opt_out1=0;         % initial optimiser output
in1=0;                  % inport input (initial)
rt=zeros(1,np);         % the control signal (scaled)
var2=0.08235;               % initial cstr output condition.
model_var=[xfwd1;xfwd2;xfwd3];      % model variables

% End initialising variables
*****************************************************************

% Start of control loop
*************************************************************************
t0 = clock;
for i=1:np

% Compute the setpoint at time t
   var1=sp(i);          % var1 = setpoints
```

```
% Sending the setpoint values to the filter
*********************************************

   % Filter output. First order discrete-time type filter.
   fout=(f*var1)+((1-f)*fout1);
   f_out(i)=fout;
   % Scale the filter output value to dscale of u (training data)
   fout_s=(fout-0.08235)/0.0164;

   rt(i)=fout_s;
   target=fout_s;              % set the scaled filter output as the target for optimiser
   mn=min(rt);
   mx=max(rt);

%*************************************************************************
************************
% Sending the setpoint values to the optimiser
*********************************************

    % Optimiser
    % Finding the optimiser output using the function fmin
      opt_out=fmin('optimisr',mn,mx,[],var1,var2,opt_out1,w,model_var,target);
      uhat(i)=opt_out;          % Scaled optimiser output put in a vector
%*************************************************************************
************************

% Sending the optimiser output to the process
*********************************************
%    in=(uhat(i)*5.6768)+99.2526; % de-scale input to process (uhat) to real value
     in=(uhat(i)*4.48)+100.26; % de-scale input to process (uhat) to real value

   % Process/cstr
   [t,x,y]=rk45('CSTR3',[0 0.1],x,[1e-3 0.0005 0.01],[0.0 in1;0.0 in;0.1 in]);
   n=length(y);
   x=x(n,:);
   cstr_op(i)=y(n);
   in1=in;
   var2=(y(n)-0.08235)/0.0164;       % scaled
```

```
%*************************************************************************
************************

% Computing the difference between setpoint and process output
*****************************

   err(i) = sp(1,i) - cstr_op(i);

   %Defining the network variables for NN updating

%     nn_var=[w;var1;model_var];
   nn_var=[w;model_var];
   % If error occurs, update the NN weights
   if err ~= 0;
     % Updating the NN if there exist error between sp(t) and y(t)
      [ynn,w,var1,var2,model_var]=nn_update(w,nn_var,var1,var2,opt_out1);
  end

%*************************************************************************
************************

% Observing the rate of change of the optimiser to reduce oscillations
*********************

   % Computing the change in optimiser output
   o_change(i)=(opt_out-opt_out1)*(opt_out-opt_out1);
   fout1=fout;          % Updating the filter output signal
   fout_s2=fout_s1;       % updating the output scaled filter signal
   fout_s1=fout_s;        % updating the output scaled filter signal
   opt_out1=opt_out;    % Updating the optimiser output
                    % Applicable only when extended cost function is used

   eer1=err.*err;

sse1(1,i)=sum(err*err');
```

```
ssc(1,i)=sum(o_change);

% Plotting of results
*************************************************************************
**
  figure(1);subplot(212);plot(sp,'k:');hold
on;plot(f_out,'b');plot(cstr_op,'k');xlabel('Setpoint and Process Output');
%  figure(1);subplot(222);plot(rt);hold on;plot(ynn,'r');xlabel('Controller Input and
Model Output');
%  figure(1);subplot(211);plot(uhat);xlabel('Optimiser Output');
  figure(1);subplot(211);plot(eer1);xlabel('SE');
%   figure(1);subplot(224);plot(err1);xlabel('Error between Process and Model');

drawnow;
end
masa=etime(clock,t0)
%  End of control loop
*************************************************************************
```

APPENDIX 2

The simulated CSTR process parameters are given below:

Tank volume, V	100ml
Specific heats, C_p, C_{pc}	1cal g^{-1} K^{-1}
Feed flowrate, q	100 l min^{-1}
Pre-exponential factor, k_o	$7.2x10^{10}$ min^{-1}
Feed concentration, C_{af}	1 mol^{-1}
Exponential factor, E/R	$9.98x10^3$ K
Feed temperature, T_f	350K
Heat of reaction, $-\Delta H$	$2.0x10^5$ cal mol^{-1}
Coolant flow rate, q_c	100 l min^{-1}
Sampling period, Δt	0.1 min
Coolant temperature, T_c	350 K
Densities, ρ, ρ_c	1000g l^{-1}
Heat transfer characteristics, hA	$7x10^5$ min^{-1} K^{-1}