5.0 Update Algorithms for Nonlinear Closed-Loop Systems

Nonlinear systems can be very difficult to control. A linear model of a nonlinear plant is often used to develop a controller for the system. The nonlinearities can cause a degradation in performance of the closed-loop system. The performance can be improved by a neural network without initially losing performance, as demonstrated in Chapter 3.

In Chapter 3, the feed-through neural network was applied with back propagation to improve the closed-loop control system for a nonlinear plant. The back propagation algorithm was not derived for a closed-loop system. It assumes an independent relationship between the inputs and the weights of the neural network. In Chapter 4, two new update algorithms were developed for FIR filters that can be used inside the closed-loop for linear systems. In this chapter, two new update algorithms are developed for neural networks inside a closed-loop system.

The update algorithms are similar to the algorithms derived in Chapter 4. However, the update algorithms were derived for a neural network consisting of a fully-connected network with a single hidden layer and a linear output layer. For the update algorithms developed, the hidden layer and the output layer have two separate equations to update the weight on the respective layers. As the number of hidden layers increases, a similar update algorithm can be derived for each new layer.

5.1 First Update Algorithm For the Neural Network

The first update algorithm is derived for minimizing the squared error between the output of the neural network and the ideal output of the neural network. The ideal output of the neural network is calculated from the reference model and the inverse model to the plant model, as seen in Figure 5.1.

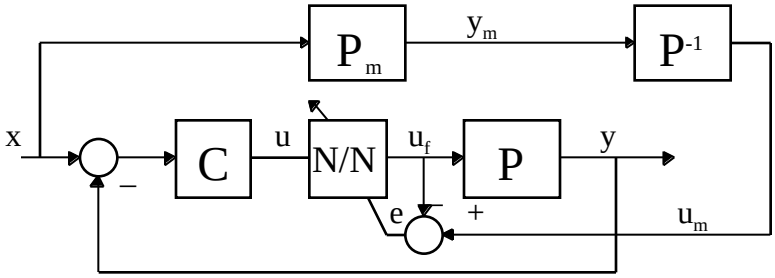

Figure 5.1 Block Diagram for First Update Algorithm

Terms used in Figure 5.1 and throughout this work are listed below.

x is the input into the system;

u is the output of the fixed gain controller;

u_f is the output of the neural network;

W_1 is the weights of the hidden layer;

W_2 is the weights of the output layer;

P is the discrete plant model;

C is the fixed gain controller;

P_m is the discrete reference model;

μ is the learning rate;

y is the output of the plant;

z is the output of the hidden layer;

f is the squashing function;

f' is the derivative of the squashing function;

y_m is the model output;

u_m is the input to the plant that would result in y matching y_m;

X is a vector of past values of x; and

Y is a vector of past values of y.

The neural network is assumed to be a fully-connected, feed-forward neural network with a single hidden layer and a single output. The output layer and one of the nodes on the hidden layer have a linear squashing function. This allows the feed-through neural network to be applied. The update algorithm is broken into two parts. There is a different algorithm for each layer, but the same error is minimized for each layer. Equation 5.1 defines the error to be minimized by this algorithm. The error is the difference between the output of the neural network and the ideal output of the neural network. The ideal output of the neural network is calculated by using the output of the reference model and the inverse of the plant model. The plant is not modeled exactly because the plant has nonlinearities that are not modeled. However, this is the plant model used to develop the fixed gain controller. Although the ideal output of the neural network is not exact, the result is that a large portion of the plant's dynamics will be accounted for. If the plant were modeled exactly, the neural network would not be required. Several equations that define key relationships can be seen in Equations 5.1 through 5.7.

$$e = u_m - u_f \qquad (5.1)$$

$$u_f = zW_2 \qquad (5.2)$$

$$z = f(W_1 U) \qquad (5.3)$$

$$u = C(x - y) \qquad (5.4)$$

$$y_m = P_m x \qquad (5.5)$$

$$u_m = P^{-1} y_m \qquad (5.6)$$

$$y = Pu_f \qquad (5.7)$$

The update algorithm for the output layer can be seen in Equations 5.8 through 5.14. The partial derivative of the error squared is taken with respect to the weights in the output layer, as seen in Equation 5.8.

$$\frac{\partial e^2}{\partial W_2} = -2e\frac{\partial u_f}{\partial W_2} \qquad (5.8)$$

Equation 5.9 is the partial derivative of the output of the neural network to the weights of the output layer. Because the output layer has a linear squashing function, the derivative of the squashing function is unity.

$$\frac{\partial u_f}{\partial W_2} = z + W_2\frac{\partial z}{\partial W_2} \qquad (5.9)$$

Equation 5.10 is the partial derivative of the output of the hidden layer with respect to the weights of the output layer. For the hidden layer, the squashing function is not assumed to be linear. The derivative of the squashing function is not unity but f'.

$$\frac{\partial z}{\partial W_2} = \frac{\partial f(W_1U)}{\partial W_2} = f'W_1\frac{\partial U}{\partial W_2} \qquad (5.10)$$

Equation 5.11 is the partial derivative of the input to the neural network, which is the output of the fixed gain controller with respect to the weights of the output layer. This is where the new update algorithm noticeably separates from the derivation of back propagation. In the derivation of the back propagation algorithm, the partial derivative of the input to the neural network with respect to the weights is zero. The reason in the new update algorithm has value for this partial derivative is because the neural network is inside the closed-loop.

$$\frac{\partial U}{\partial W_2} = \frac{\partial C(x-y)}{\partial W_2} = -\frac{\partial Cy}{\partial W_2} = -C\frac{\partial y}{\partial W_2} \qquad (5.11)$$

Equation 5.12 is the assumed form for the plant.

$$y(k) = -b_1 y(k-1) - b_2 y(k-2) .... - b_l y(k-l) + a_0 u(k) + a_1 u(k-1) + ... \qquad (5.12)$$

Equation 5.13 is the partial derivative carried through the plant. A key assumption is made about the plant: the input to the plant does not directly feed through; the term a_o is zero. With this assumption, the partial derivative of the output of the plant with respect to the weights can be calculated from values of the past iterations.

$$\frac{\partial y(k)}{\partial W_2} = -b_1 \frac{\partial y(k-1)}{\partial W_2} - b_2 \frac{\partial y(k-2)}{\partial W_2} ... + a_0 \frac{\partial u_f(k)}{\partial W_2} + a_1 \frac{\partial u_f(k-1)}{\partial W_2} \qquad (5.13)$$

The update algorithm can be seen in Equation 5.14. The weights for the output layer can be adjusted during each iteration.

$$W_2 = W_2 + 2\mu e \frac{\partial u_f}{\partial W_2} \qquad (5.14)$$

This is the derivation for the update algorithm of weights on the hidden layer. The partial derivative of error squared with respect to the weights of the hidden layer can be seen in Equation 5.15.

$$\frac{\partial e^2}{\partial W_1} = -2e\frac{\partial u_f}{\partial W_1} \qquad (5.15)$$

Equation 5.16 is the partial derivative of the output of the neural network with respect to the weights on the hidden layer. Because the squashing function is linear for the output layer, the derivative of the squashing function is unity.

$$\frac{\partial u_f}{\partial W_1} = W_2 \frac{\partial z}{\partial W_1} \qquad (5.16)$$

Equation 5.17 is the partial derivative of the output of the hidden layer with respect to the weights on the hidden layer. Because the squashing function is not linear for the hidden layer, the derivative of the squashing function, f', is not unity.

$$\frac{\partial z}{\partial W_1} = \frac{\partial f(W_1 U)}{\partial W_1} = f'(U + W_1 \frac{\partial U}{\partial W_1}) \qquad (5.17)$$

Equation 5.18 is the partial derivative of the input to the neural network, which is the output to the fixed gain controller with respect to the weights of the hidden layer.

$$\frac{\partial U}{\partial W_1} = \frac{\partial C(x - y)}{\partial W_1} = -\frac{\partial Cy}{\partial W_1} = -C\frac{\partial y}{\partial W_1} \qquad (5.18)$$

Equation 5.19 is the assumed form of the plant.

$$y(k) = -b_1 y(k-1) - b_2 y(k-2) \ldots . - b_l y(k-l) + a_0 u(k) + a_1 u(k-1) + \ldots . \qquad (5.19)$$

Equation 5.20 is the partial derivative of the plant with respect to the weights on the hidden layer. The feed-through term on the plant's input is assumed to be zero. The value of the partial derivative of the plant can be calculated using values of previous iterations.

$$\frac{\partial y(k)}{\partial W_1} = -b_1 \frac{\partial y(k-1)}{\partial W_1} - b_2 \frac{\partial y(k-2)}{\partial W_1} \ldots + a_0 \frac{\partial u_f(k)}{\partial W_1} + a_1 \frac{\partial u_f(k-1)}{\partial W_1} \qquad (5.20)$$

The equation to update the weights of the hidden layer can be seen in Equation 5.21.

$$W_1 = W_1 + 2e\frac{\partial u_f}{\partial W_1} \qquad (5.21)$$

With the derivation of the update algorithm for each layer, the algorithm is applied to two examples, one stable and one unstable. The examples are the same nonlinear examples used in Chapter 3 with the feed-through neural network with back propagation.

5.2 Results of the First Update Algorithm

The update algorithm is applied to two different nonlinear plants. The first plant is stable and the second plant is unstable. Figure 5.2 is the block diagram for the implementation of the first algorithm with the estimator/regulator controller. The algorithm does not have a problem with any type of discrete controller because a dummy controller can be created to calculate the partial derivatives in order to calculate all of the variables needed to update the weights on each layer.

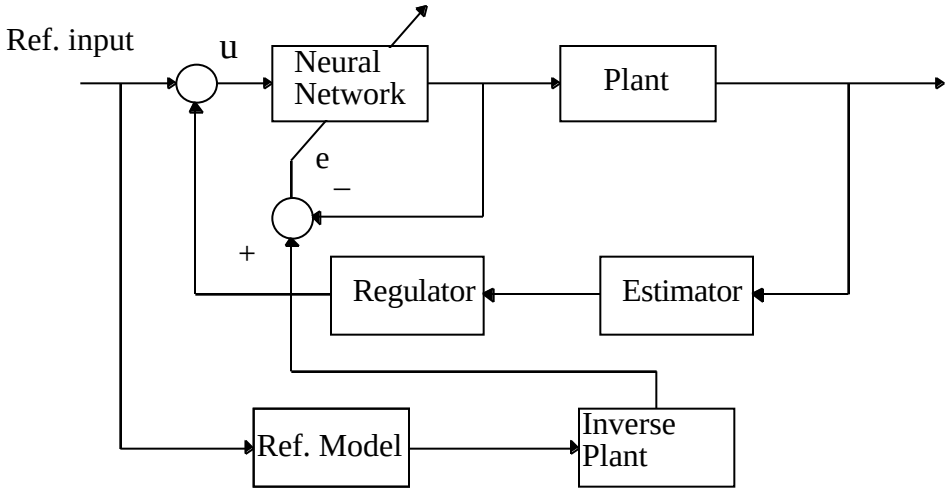

Figure 5.2 Block Diagram for Nonlinear System Example

The stable nonlinear plant has the first update algorithm applied to it with a feed-through neural network. The feed-through neural network has a gain of unity and allows the fixed-gain controller to initially control the closed-loop system as if the neural network did not exist. The first update algorithm reduces the amount of effects due to the plant's dynamics that can inhibit the convergence of the neural network. By using a-priori knowledge of the system, the first update algorithm converges the weights of the neural network very quickly. For this example, the weights converge within 4000 iterations. This result is comparable to the back propagation results, which took 300,000 iterations to converge. The improved performance was achieved with a single convergence of the neural network. The first update algorithm reduces the mean squared error of the closed-loop system from 3.27×10^{2} to 3.27×10^{4} , a 99% reduction of mean squared error for the step input. The results of the step input with and without the neural network can be seen in Figure 5.3.

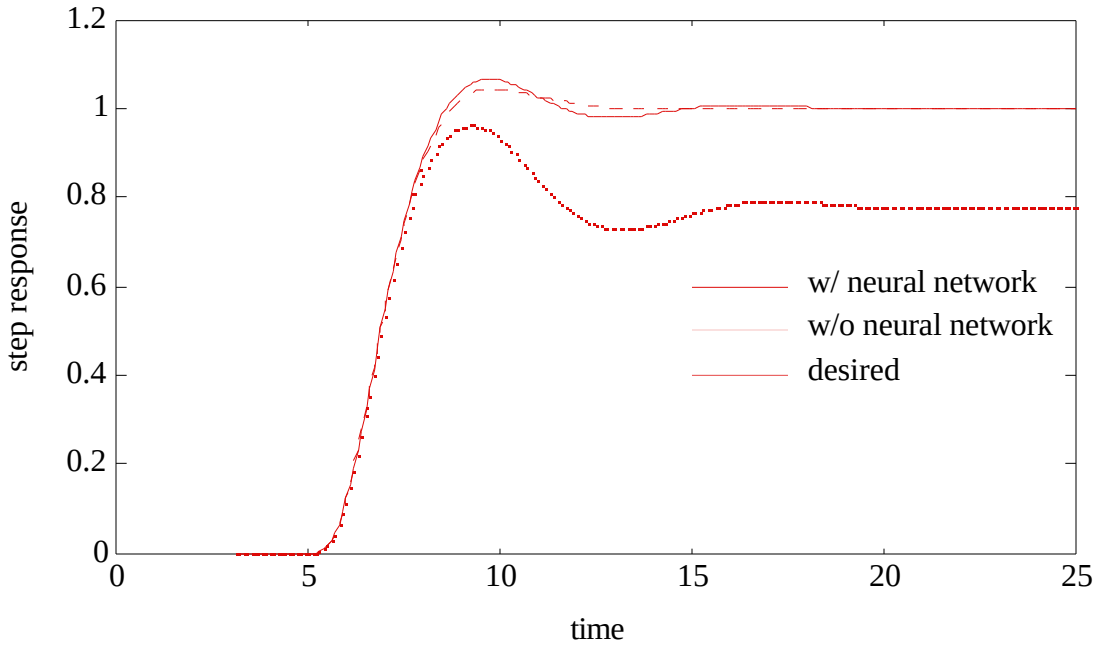

Figure 5.3 Results of Stable Example to a Step Input

The second example to which the first update algorithm was applied is the unstable plant used in Chapter 3. The block diagram of the plant can be seen in Figure 5.2. The first update algorithm is applied to the example with a feed-through neural network, which gives an initial performance identical to the fixed gain controller without the neural network. The neural network greatly improves the performance of the unstable plant. The performance of the closed-loop system is achieved in a single convergence of the weights. The closed-loop system is trained with white noise ranging from +/- 1.5. The mean square error is reduced from 1.16×10^2 to 2.06×10^3, a reduction of 82%. As seen in Figure 5.4, the response to the step input has been improved. The system converges within 4000 iterations, compared to the results with back propagation that took 400,000 iterations to converge.

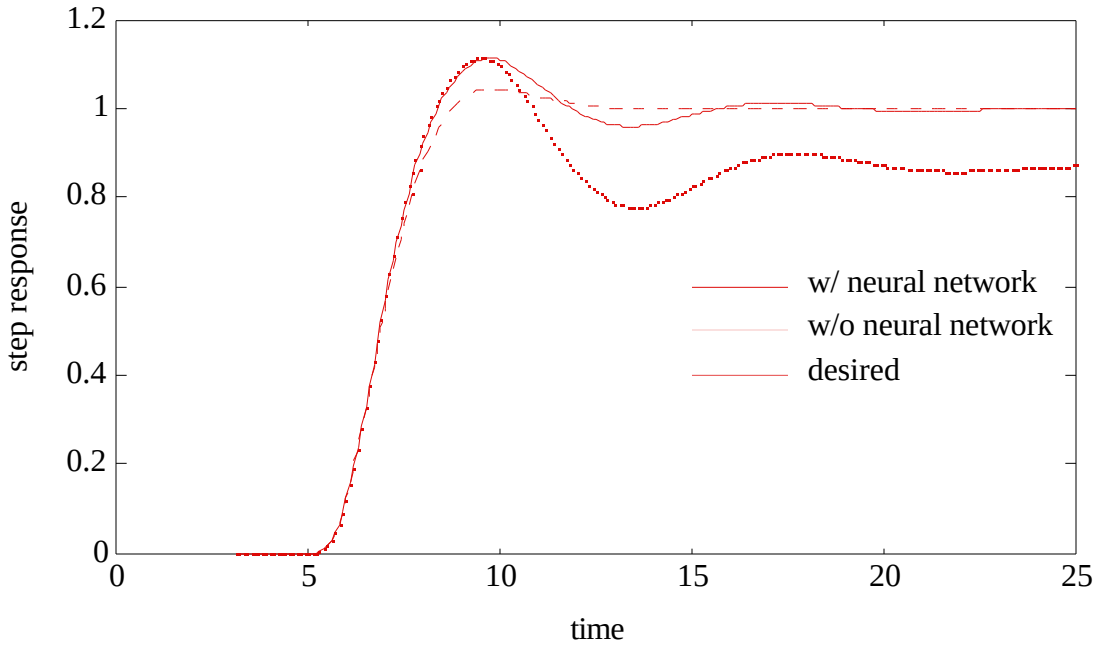

Figure 5.4 Results of Unstable Example for a Step Input

The first update algorithm greatly improves the performance of the closed-loop system. The algorithm reduces the time to converge down to less than 2% of that required by back propagation. The reduction comes from the a-priori knowledge used by the algorithm to decrease the interference of the plant's dynamics with the convergence process.

The first update algorithm has two primary weaknesses. First, an inverse to the plant's model is needed to create the ideal neural network output; however, the inverse model may not be stable. Second, the error between the output of the neural network and the ideal output of the neural network is not the variable of interest; instead, the variable of interest is the difference between the output of the plant and the output of the reference model.

A second update algorithm is developed specifically to address these two weaknesses. The algorithm minimizes the error between the output of the plant

and the output of the reference model. The second algorithm does not require the ideal output of the neural network; thus, it does not require an inverse plant model.

5.3 Second Update Algorithm for a Neural Network

The second update algorithm is derived to minimize the output of the plant and the output of the reference model, as seen in Equation 5.22. The algorithm was developed to update the weights of the neural network within the closed-loop, as seen in Figure 5.5. The neural network is assumed to have a single hidden layer and a single output from an output layer that has a linear squashing function. The nodes on the hidden layer all have nonlinear squashing functions except for one. This neural network configuration allows a feed-through neural network to be used. An algorithm is developed for both the hidden layer and the output layer. The derivation of the second update algorithm is very similar to the derivation of the first update algorithm.

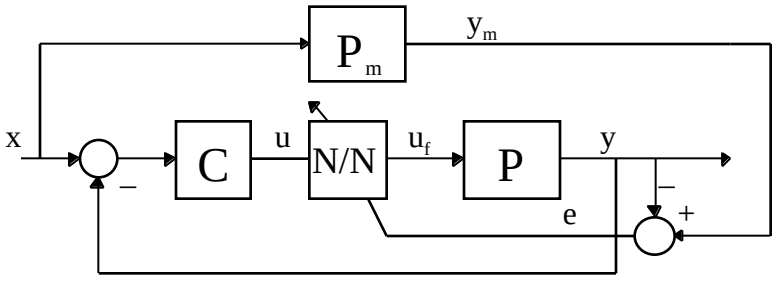

Figure 5.5 Block Diagram for Second Update Algorithm

$$e = y_m - y \qquad (5.22)$$

Equation 5.23 is the partial derivative of the error with respect to the weights of the output error.

$$\frac{\partial e^2}{\partial W_2} = 2e\frac{\partial (y_m - y)}{\partial W_2} \quad (5.23)$$

In Equation 5.24, y_m is eliminated because it is not dependent on the weights.

$$\frac{\partial e^2}{\partial W_2} = -2e\frac{\partial y}{\partial W_2} \quad (5.24)$$

Equation 5.25 is the assumed form of the plant.

$$y(k) = -b_1 y(k-1) - b_2 y(k-2) .... - b_l y(k-l) + a_0 u(k) + a_1 u(k-1) + ... \quad (5.25)$$

Equation 5.26 is the partial derivative of the plant's output with respect to the weights of the output layer. The direct feed-through term is assumed to be zero. This allows the calculation $\frac{\partial y(k)}{\partial W_2}$ with information from previous iterations.

$$\frac{\partial y(k)}{\partial W_2} = -b_1\frac{\partial y(k-1)}{\partial W_2} - b_2\frac{\partial y(k-2)}{\partial W_2} ... + a_0\frac{\partial u_f(k)}{\partial W_2} + a_1\frac{\partial u_f(k-1)}{\partial W_2} \quad (5.26)$$

Equation 5.27 is the partial derivative of the neural network's output with respect to the output layer. The squashing function is linear for the output layer, thus making the derivative of the squashing function unity.

$$\frac{\partial u_f}{\partial W_2} = z + W_2\frac{\partial z}{\partial W_2} \quad (5.27)$$

Equation 5.28 is the partial derivative of the output of the hidden layer with respect to the output layer. The squashing function is not linear; thus, the derivative of the squashing function is not unity.

$$\frac{\partial z}{\partial W_2} = \frac{\partial f(W_1 U)}{\partial W_2} = f' W_1 \frac{\partial U}{\partial W_2} \quad (5.28)$$

Equation 5.29 is the partial derivative of the input to the neural network with respect to the weights of the output layer.

$$\frac{\partial U}{\partial W_2} = \frac{\partial C(x - y)}{\partial W_2} = -\frac{\partial Cy}{\partial W_2} = -C\frac{\partial y}{\partial W_2} \qquad (5.29)$$

Equation 5.30 is the weight update algorithm for the output layer. The weight update algorithm does not require the inverse model of the plant and minimizes the difference between the plant's output and the reference model's output, which is the primary variable of interest.

$$W_2 = W_2 + 2e\frac{\partial y}{\partial W_2} \qquad (5.30)$$

The second part of the update algorithm is the derivation of the update algorithm for the hidden layer. The partial derivative of the error squared with respect to the weights of the hidden layer can be seen in Equation 5.31.

$$\frac{\partial e^2}{\partial W_1} = -2e\frac{\partial y}{\partial W_1} \qquad (5.31)$$

The assumed form of the plant can be seen in Equation 5.32.

$$y(k) = -b_1 y(k-1) - b_2 y(k-2) \ldots . - b_l y(k-l) + a_0 u(k) + a_1 u(k-1) + \ldots . \qquad (5.32)$$

Equation 5.33 is the partial derivative of the plant's output with respect to the weights of the hidden layer. The derivative of direct feed-through term is

assumed to be zero. The calculation of the partial derivative can be done with values from previous iterations.

$$\frac{\partial y(k)}{\partial W_1} = -b_1 \frac{\partial y(k-1)}{\partial W_1} - b_2 \frac{\partial y(k-2)}{\partial W_1} \ldots + a_0 \frac{\partial u_f(k)}{\partial W_1} + a_1 \frac{\partial u_f(k-1)}{\partial W_1} \qquad (5.33)$$

Equation 5.34 is the partial derivative of the output of the neural network with respect to the weights of the hidden layer.

$$\frac{\partial u_f}{\partial W_1} = W_2 \frac{\partial z}{\partial W_1} \qquad (5.34)$$

Equation 5.35 is the partial derivative of the output of the hidden layer with respect to the weight of the hidden layer.

$$\frac{\partial z}{\partial W_1} = \frac{\partial f(W_1 U)}{\partial W_1} = f'(U + W_1 \frac{\partial U}{\partial W_1}) \qquad (5.35)$$

Equation 5.36 is the partial derivative of the input of the neural network with respect to the weights of the hidden layer.

$$\frac{\partial U}{\partial W_1} = \frac{\partial C(x-y)}{\partial W_1} = -\frac{\partial Cy}{\partial W_1} = -C\frac{\partial y}{\partial W_1} \qquad (5.36)$$

Equation 5.37 is the weight update algorithm for the hidden layer.

$$W_1 = W_1 + 2e\frac{\partial y}{\partial W_1} \qquad (5.37)$$

The second update algorithm can be applied to a nonlinear system to improve the performance of a closed-loop control system. The examples, stable and

unstable nonlinear plants (from Chapter 3 and Section 5.2), are again used to verify the performance of the second algorithm.

5.4 Results of Second Update Algorithm

The second update algorithm for both the output layer and the hidden layer are implemented on the same stable and unstable nonlinear plants as the first update and the back propagation algorithms. The feed-through neural network is used with the second algorithm to increase the performance of the closed-loop system, which were originally done in Chapter 3. The closed-loop system is designed with an estimator/regulator as the control system. The plant has an unmodeled nonlinearity, which is a saturation between two pairs of poles. The block diagram of the overall closed-loop control system can be seen in Figure 5.7.

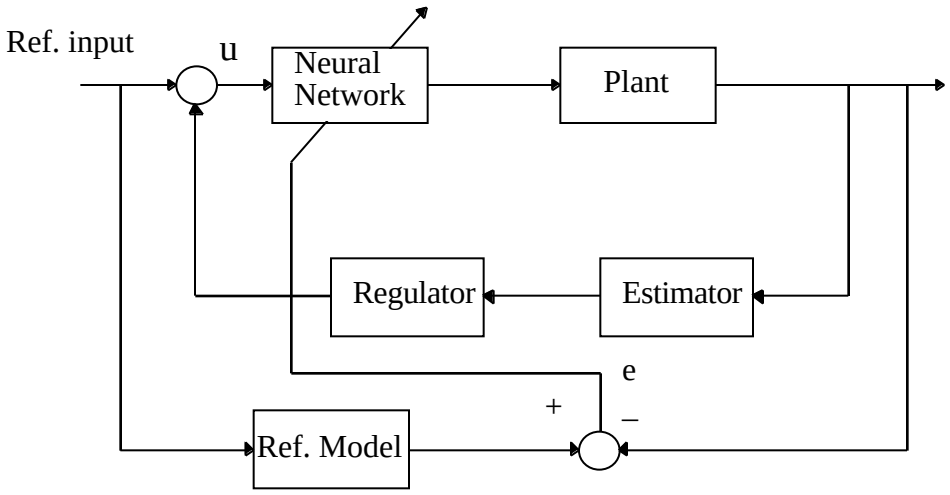

Figure 5.6 Block Diagram for the Specific Example

The stable example has the second update algorithm used on it to improve the performance with the unmodeled nonlinearity. The performance of the closed-loop system is greatly improved for the step input, as seen in Figure 5.8. The mean squared error is reduced from 3.27×10^{2} to 2.74×10^{4}, a 99% reduction.

The weights of the neural network converge within 5000 iterations, which is a vast improvement over back propagation. Back propagation takes more than 50 times longer to converge. The results are achieved with a single convergence to the weights from their initial feed-through configuration. The solid line in Figure 5.8 shows the performance of the closed-loop control system after the neural network converges. The dotted line represents the performance of the closed-loop system without the neural network and the performance before the neural network starts to converge.

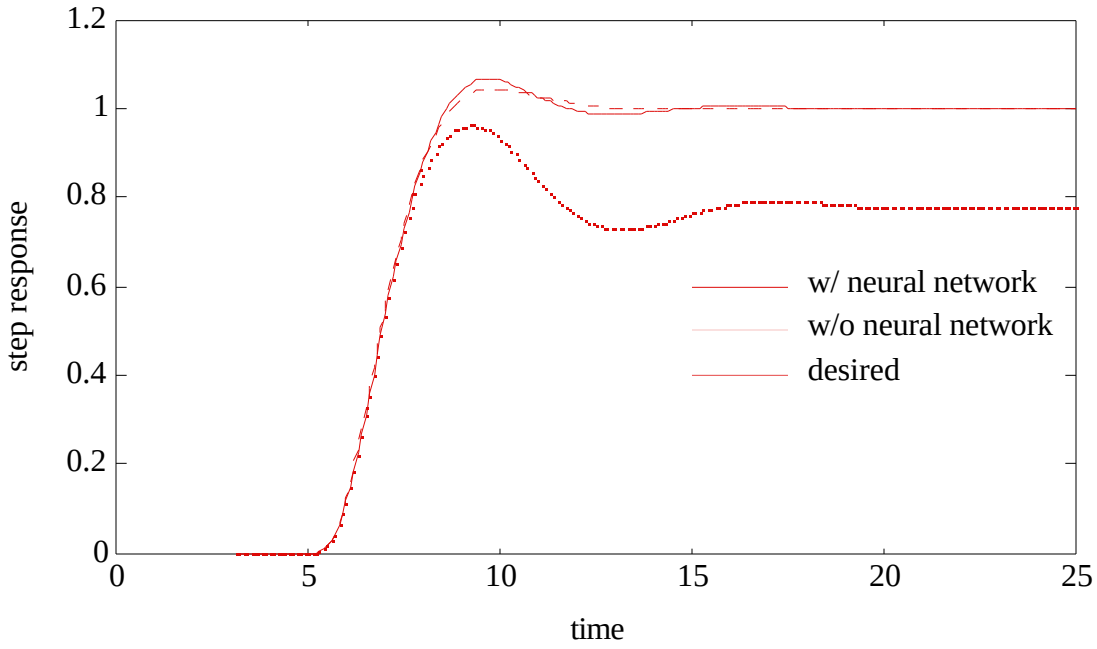

Figure 5.7 Results of the Stable Example for a Step Input

The unstable plant has the second update algorithm applied to it. The results of the step input to the system are shown in Figure 5.9. The performance of the system is greatly increased. The weights are converged once from the feed-through initial position. White noise is used to train the neural network. The mean squared error is reduced from 1.16×10^{2} to 1.83×10^{3}, and the convergence of the weights took 5000 iterations, much faster than the back propagation algorithm.

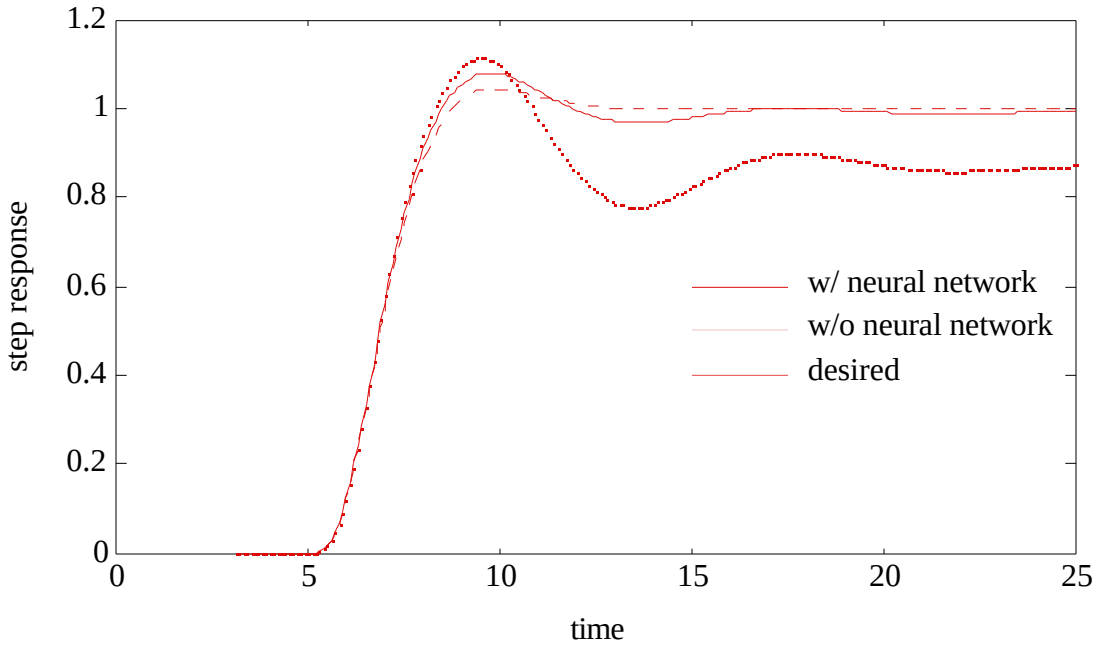

Figure 5.8 Results of the Unstable Example for a Step Input

The second update algorithm works well on the stable and unstable plants. The weights converge quickly to increase the performance of the closed-loop control system. The results show that the second update algorithm improves the performance of the closed-loop control system of a nonlinear plant.

5.5 Summary

The success of the two new update algorithms can be seen in the results of the step input. Both algorithms greatly improve the performance of the closed-loop system. The comparison of the three different update algorithms, back propagation and the two new ones, can be seen in Figure 5.10 for the stable plant and Figure 5.11 for the unstable plant. The comparison of the three mean squared errors can be seen in Table 5.1.

Table 5.1 Comparison of Mean Square Error for Step Input

	Stable	Unstable
Initially	0.0327461	0.0115941
Back Propagation	0.0002855	0.0018523
First Algorithm	0.0003268	0.0020641
Second Algorithm	0.0002737	0.0018273

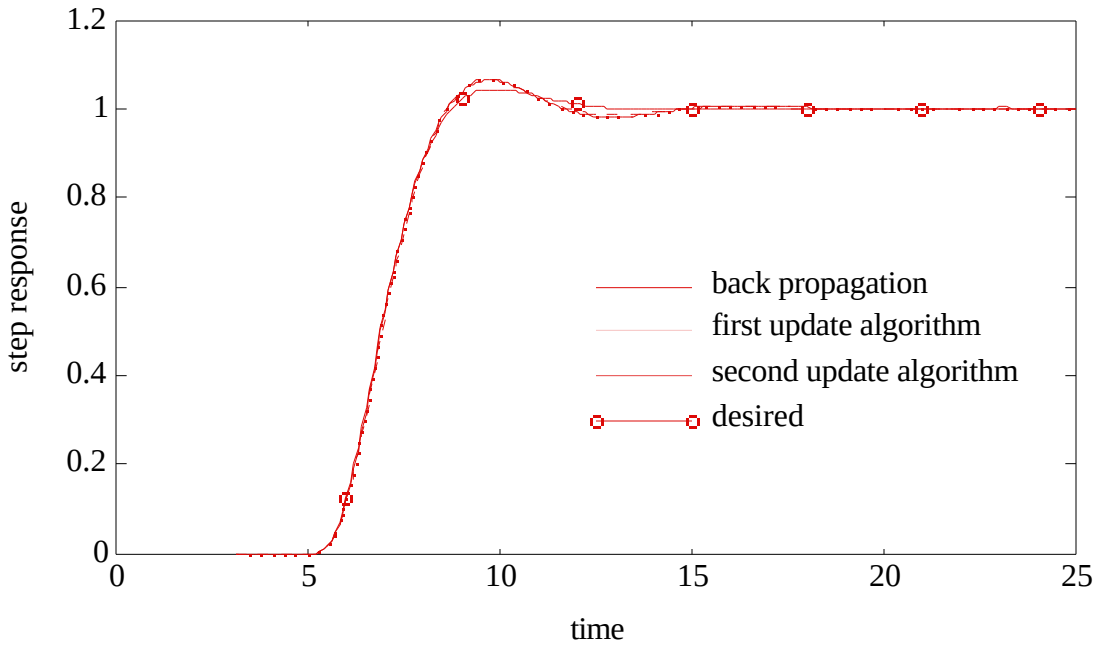

Figure 5.9 Summary of History for the Stable Example

The results of the converged systems are very similar. There is only a negligible difference between the three methods. In Figure 5.10, the converged systems for the stable plant cannot be distinguished from each other. Each method results in a 99% reduction in the mean square error. In Figure 5.11, the converged systems for the unstable plants have a small distinction between each other compared to a marked improvement of the overall system.

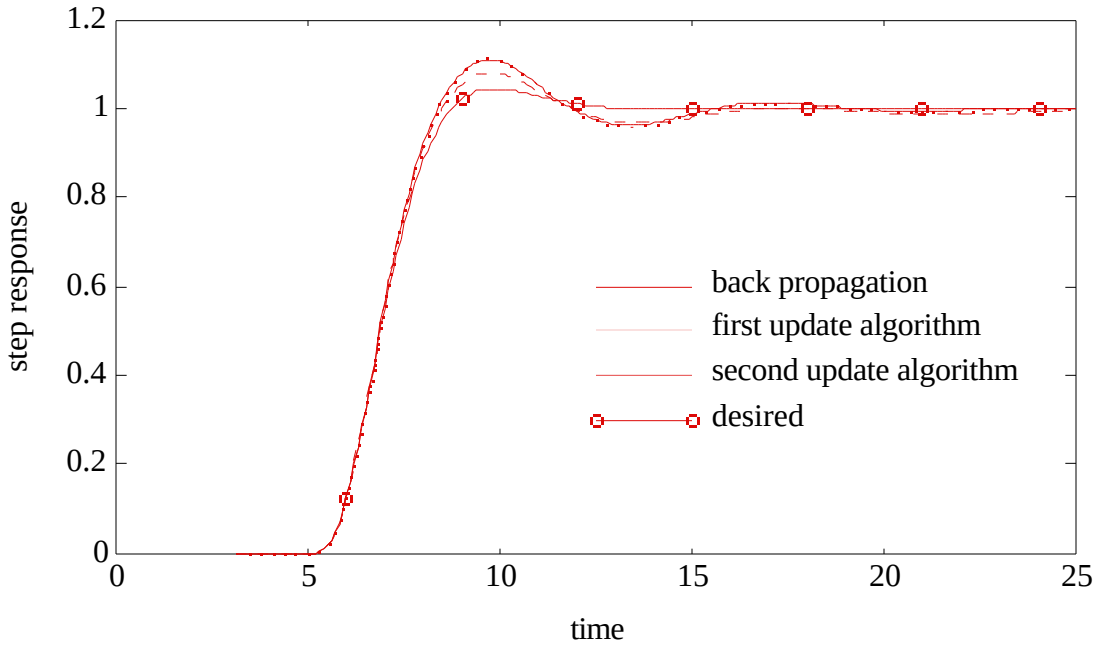

Figure 5.10 Summary of History for the Unstable Example

The second algorithm has the least amount of mean square error for the unstable example. Also, it has the worst performance of the three algorithms. This is the only case in which the results are different. The second algorithm most reduces the amount of overshoot of the system. However, it left a small amount of steady state bias. The steady state bias is a much larger problem than the overshoot. The second algorithm increases the performance of the closed-loop system, but the other two algorithms converge to a superior solution.

The three algorithms work very well. There is very little distinction between the final results of the three algorithms. The back propagation algorithm is much less computationally intensive than the two new algorithms. Because the results are relatively the same, it is difficult to see the advantage of the two new algorithms. The two algorithms are designed to reduce the problems associated with using neural networks for control. The plant's dynamics

inherently causes convergence problems. This can greatly reduce the rate of convergence.

In order to compare the convergence of the different methods, a training set of uniform noise is created. The training set is a million points between +/- 1.5. A feed-through neural network with the exact same configuration and learning rate is used. This establishes the same conditions for each of the three algorithms. The results can be seen in Figure 5.12. From that figure, it is easy to see the advantage of the two new algorithms. For the stable example, back propagation takes approximately 300,000 iterations to converge. This is compared with the convergence of the two new algorithms, which can better be seen in Figure 5.13, with the shortened scale. The first update algorithm converges for the stable example within 4000 iterations; the second update algorithm converges within 5000 iterations. Both algorithms stay stable throughout the entire training set.

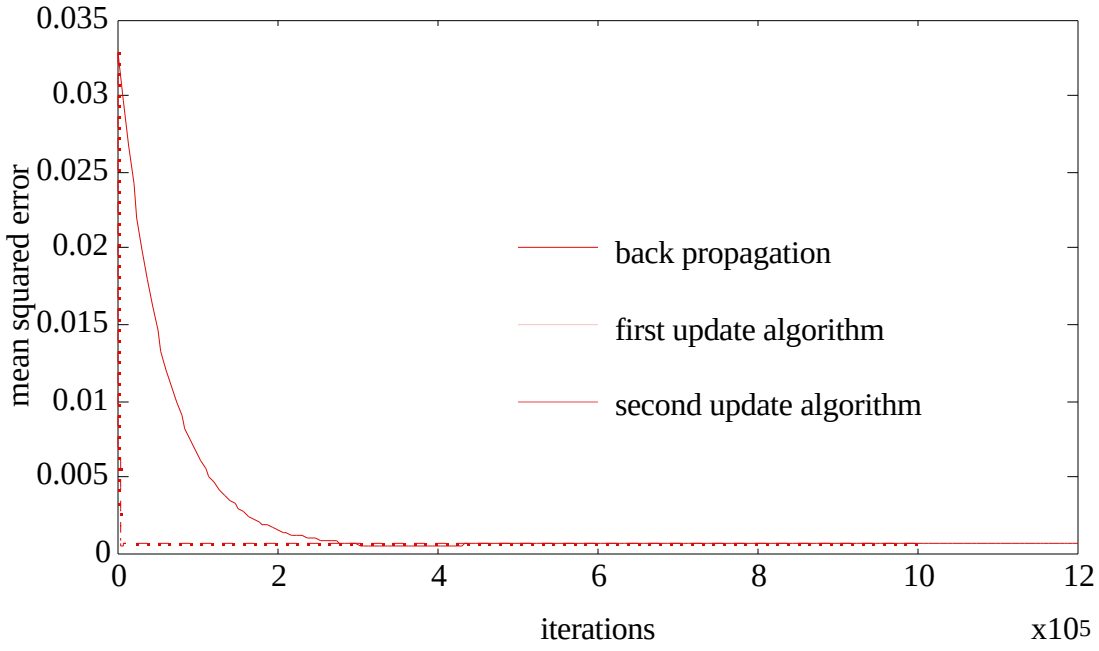

Figure 5.11 Convergence History of Stable Example

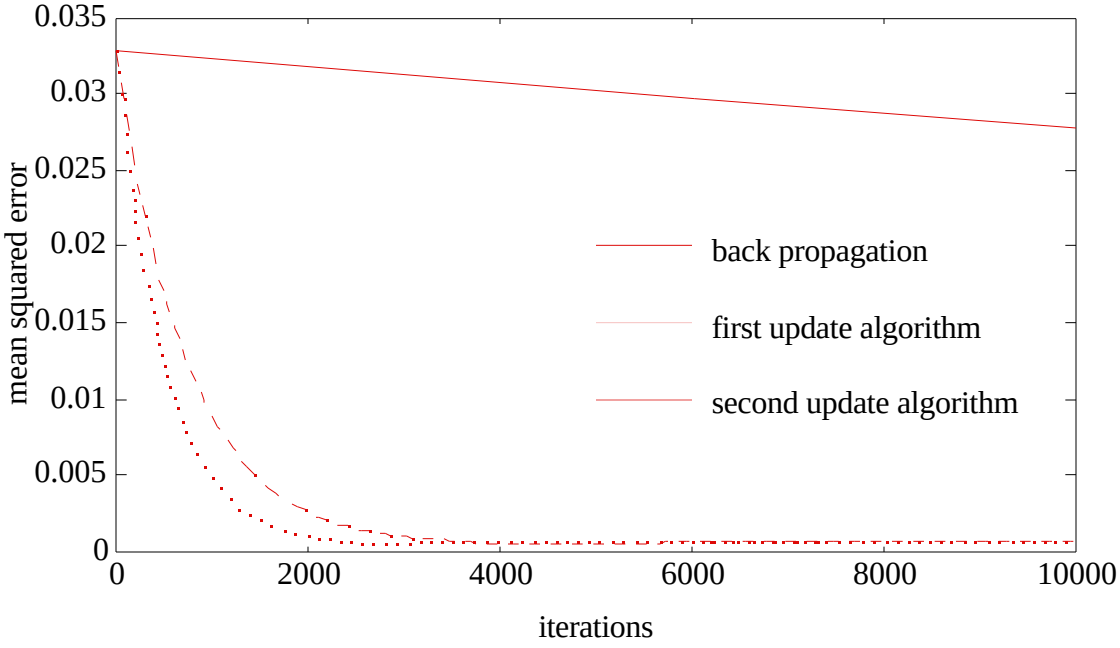

Figure 5.12 Convergence History of Stable Example Rescaled

Applying the same training set to the unstable example, the results are similar to that of the stable example. The performance of the three algorithms are all very similar. However, the two new algorithms have converged in a much shorter number of iterations, as seen in Figure 5.14. In Figure 5.15, the greatly decreased convergence time is very evident with the shortened scale.

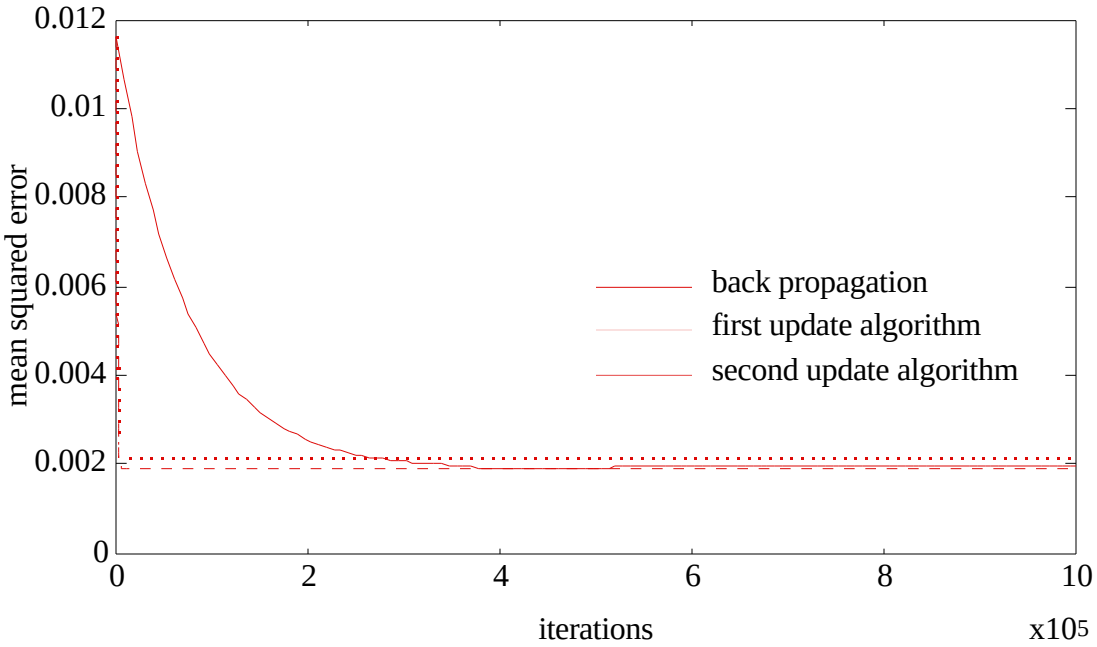

Figure 5.13 Convergence History for Unstable Example

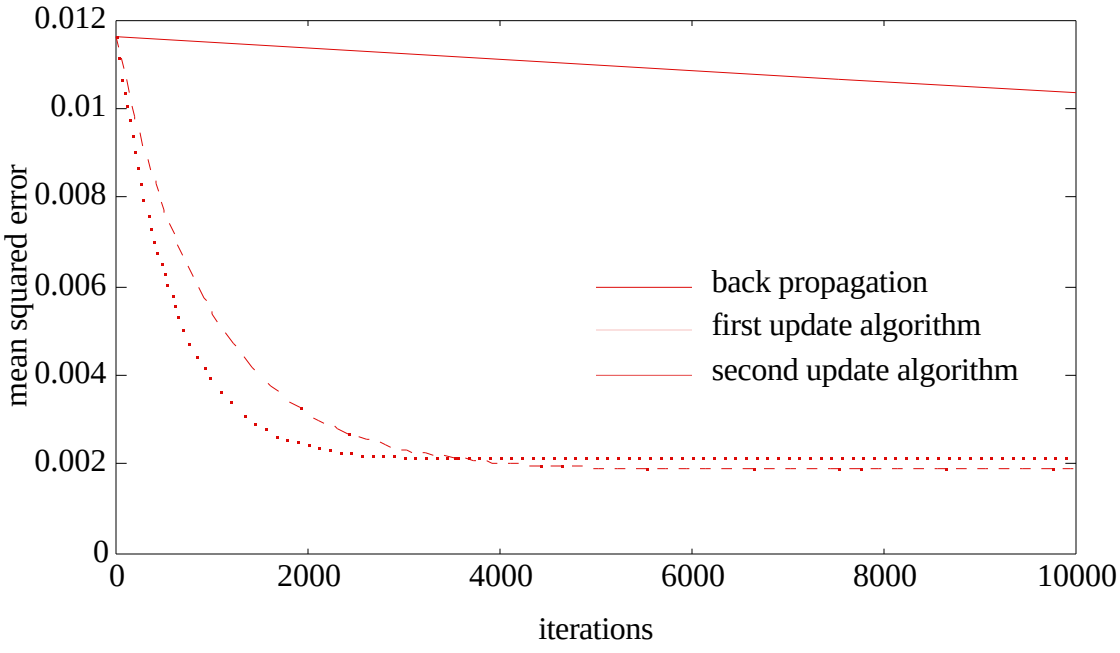

Figure 5.14 Convergence History for Unstable Example Rescaled

The two new update algorithms have very similar results to back propagation. However, the amount of time required to converge is much less than that of the back propagation algorithm. This result is not surprising because the back propagation algorithm is not designed to operate inside the closed-loop. Back propagation is also not a model-based algorithm. If a priori knowledge of the system is available, the two new algorithms capitalize on this information and reduce the convergence time.

In the next chapter, the three algorithms are to be applied to an example with a greater unknown nonlinearities. An example of a boiler plant that actually models the transport delays of the system will use the three algorithms to improve the closed-loop system. Each algorithm will be evaluated on its results.