Path Following Using A Learning Neural Network

NHH Mohamad Hanif

Supervisor: Prof. Robin Sharp

This report is submitted in partial fulfillment of the requirements for the Degree of Master of Science (MSc) in Control Systems, University of London, and the Diploma of Imperial College.

Control and Power Group,
Department of Electrical and Electronic Engineering,
Imperial College London,
University of London

September 2004

ABSTRACT

This thesis is a development of previous works done by [2] on capability of neural controller to efficiently track prescribed paths. Equipped with knowledge on optimal preview control obtained from [1], the initial weights of linear and nonlinear neural controller are initialized to the optimal gains. The implemented neural controller will in turn minimize a performance index, which includes the lateral and attitude angle errors of vehicle models with respect to the paths.

The thesis differs from [2] in a sense that different types of neural controller are established to achieve a better path following accuracy. Two algorithms, gradient descent and quasi-Newton which utilize a batch training method, are introduced as comparison to the gradient descent method that incorporates the online (or incremental) training method. The class of learning (whether good or bad) of the neural controllers is evaluated from the obtained percentage of average weight change, maximum path and yaw attitude angle errors as well as the maximum steering wheel angle. The behaviors of learning rates and updated weights are given special attention in this thesis. To conduct the specified works, the MATLAB programs written by [2] have been extended and modified.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Throughout the whole period of the project execution, many had provided immeasurable amount of guidance, ideas and help. Without the assistance from these people, this research would have not been successful. The author would like to express her utmost gratitude to the following individuals and members:

- *Prof. Robin Sharp,* Supervisor of the project, for giving such a good guidance and advises regarding the project and difficulties in project works

- *Mr. Pierre Dandré,* for sharing information on his previous works in further detail

- *Miss D.S. Laila,* for outstanding explanation on optimal control theory

- Lastly, but never the least, to my beloved parents and siblings, for the prayers, for moral and emotional support and for being there when needed most

List of Figures

List of Tables

List of Appendices

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Many researches on automated car control have been conducted during these past few years. Quite a number of efforts have been concentrated on the capability of the self-guided vehicles to accurately follow various types of paths. A vital factor in realizing the aim is steering control.

Various approaches have been carried out for this purpose, including optimal preview control and neural network. Optimal preview control is capable to portray the driver's vision of the path and process the knowledge so that the vehicle can follow the path as accurately as possible. A controller that utilizes the technology of neural network has the ability to 'learn' from past errors and adjust the network to obtain specific target output. In other words, provided that suitable weights are used, a neural controller will achieve a more precise path following.

This study aims to develop a neural network controller that could control both linear and nonlinear vehicle models to follow prescribed paths with the smallest errors. Different types of neural controller are introduced for comparison purposes. The behaviours of the learning rates and updated weights are also investigated.

1.2 Brief Outline on Previous Works

This thesis has been conducted with reference to works by Sharp and Valtetsiotis [1] on optimal preview control and Dandré [2] on the upgraded performance of the optimal preview control with the use of neural network.

The first reference is about representation of driver's vision, in which road and linear vehicle information (in discrete time equation) are combined and assessed by the linear quadratic cost function. The implemented optimal control minimizes the cost (lateral errors and attitude angles relative to the path) according to its priority, which is path following.

The second reference shows comparison of weights initialized to zero, and weights obtained as optimal gains from works of Sharp and Valtetsiotis [1] to implement single and multi-layered neural controller. The algorithm used is gradient descent, and the training mode is identified as online (or incremental) training. Tracking simulations are done on linear and nonlinear car models.

1.3 Outline of Thesis

Chapter 2 is a review on neural networks; its definition, training methods (batch or online) and algorithms (gradient descent and Quasi-Newton).

Chapter 3 is a review on previous works by Sharp [1] and Dandré [2]. This chapter summarizes the linear and nonlinear car models, path models as well as the optimal preview controller, which has the ability to drive a linear car model on simulated paths.

Chapter 4 outlines the implementation of a linear neural network that could control the linear car to accurately follow the simulated paths. Two types of training are introduced: online training and batch training. Two algorithms are introduced: gradient descent method and Quasi-Newton method. The learning ability of the neural controller is judged by the average percent weight change by learning and the maximum errors obtained through the MATLAB simulations.

Chapter 5 revolves around implementation of a nonlinear neural network that is designed to control nonlinear car to follow prescribed paths. Researches and simulations for the nonlinear network are conducted similarly as for the linear network (Chapter 4).

Chapter 6 concludes the study and outlines recommendations for future works.

1.4 Contribution of the Thesis

- Proves that the training of neural network for more than one epochs would increase the controller performance in most situation
- Shows that the batch training, namely using the gradient and quasi-Newton methods, could be implemented to several situations, in which better outputs are achieved with a shorter training time
- Confirms that the behaviours of the learning rates play a vital role in shaping the behaviours of the updated weight; in which the learning rates will slowly reduced towards zero, leading small variations of updated weights after several epochs

CHAPTER 2

REVIEW ON NEURAL NETWORKS FOR CONTROL

This chapter is an overview on basic structure of neural network, its training schemes and algorithms, which are significant factors in implementing the neural controller for the simulated paths and car models.

2.1 BACKGROUND OF NEURAL NETWORK

Neural networks, inspired by biological nervous system, are composed of simple elements operating in parallel. Demuth and Beale [3] described, "Neural networks are adjusted, or trained, so that a particular input leads to a specific target output". The particular situation is as shown in Figure 2.1:

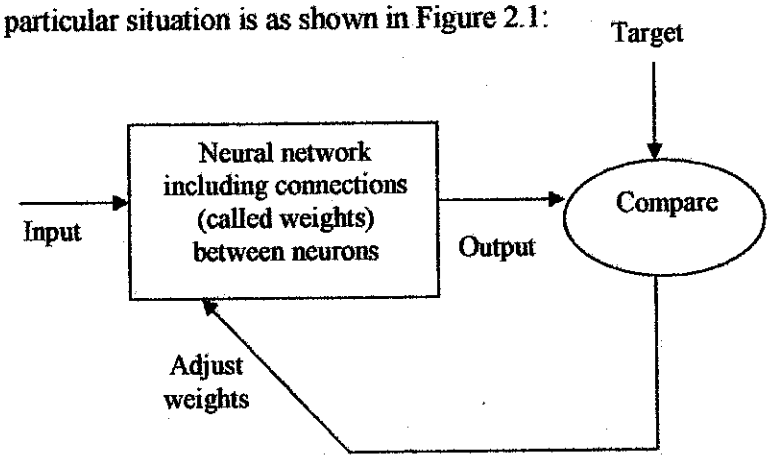

Target
Neural network including connections (called weights) between neurons
Input
Output
Compare
Adjust weights

Figure 2.1: *Adjustment of neural network to obtain specific target output*

Neural Network performs two major functions: *Learning* and *recall. Learning* is the process of adapting the connections in a neural network to produce a desired output vector in response to a stimulus vector presented in the input buffer. *Recall,* on the other hand, is the process of accepting input stimulus and producing output response in accordance with the network weight structure.

2.2 NEURAL NETWORK TRAINING

Neural networks could produce desirable outputs by having sufficient training. Commonly the networks are adjusted, or trained so that a particular input leads to a specific target output.

There are two different styles of training, which are *incremental training* and *batch training*. The training styles differ in terms of how the weights and biases are adjusted.

- *Online Training*

Online training updates weights and biases as each input is presented to the network. By setting a value of network learning rate, the weights will change at each subsequent time step (instance). Thus, weights are updated more than once per entire presentation of training data (epoch).

- *Batch Training*

According to Bersetkas and Tsitsiklis [5], batch algorithm is a conventional numerical optimization technique. By implementing batch training, weights and biases are accumulated over an epoch before being updated. Thus, in each epoch, the weights are only updated once. Another alternative (but similar mode of operation) to batch training is mini-batch training. In this case, weight changes are accumulated over some number of instances before being updated.

2.2 NEURAL NETWORK ALGORITHM

The most commonly used neural network learning algorithm is back propagation. The term refers to the manner in which the gradient is computed for nonlinear multilayer networks [6]. Standard back propagation is a gradient descent algorithm, in which the network weights are moved along the negative of the gradient of the performance function.

This algorithm has different variations based on the standard optimization techniques. The variations include the gradient descent, conjugate gradient descent, Newton, Quasi-Newton and Levenberg-Marquardt method. The applications of these algorithms rely on the scale of the network to be used. Gradient descent method is typically for a large scale network, conjugate direction is for a medium scale, Quasi-Newton and Levenberg-Marquardt (preferred for low residual regression problems) for small scale while Newton method is for a tiny scale network [7]. Two methods were used for this project, and are described in this section. The methods are gradient descent and Quasi-Newton method.

2.2.1 Gradient Descent Method

In neural network, the gradient descent learning is applied to determine network weights that minimize error functions. The two parameters (weight and error functions) create an error surface. This algorithm usually initializes at a commonly random point in the weight space and points along the line of steepest descent until a minimum in the error surface is found. As the sequences of the points reaching to the minimum, the changing rate from the previous to next points decreases.

This particular manner is due to the formulation of the gradient descent learning itself:

$$\Delta w = -\gamma \frac{\partial J}{\partial w} \qquad (2.1)$$

where w is the weighting vector, J is the performance and γ is the learning rate. The negative sign implies that the gradient descent is approximated by taking small but finite steps in the direction of steepest descent. As soon as the weights just start to change in the direction of the gradient at the measured point, the true gradient itself will start to change. [8]. Thus, as the algorithm progresses, the learning rate will be getting smaller and approaches zero.

A gradient descent algorithm by itself has a slow response. To increase the rate of response, momentum term is combined with the basic algorithm. This combination results in movement in fixed direction. Thus, if several steps are pointed towards the same direction, the rate of response of the algorithm will increase.

Another mode of the gradient descent algorithm that is applied in this research is gradient descent with adaptive learning rate back propagation. Without adaptive learning, the learning rate is kept constant throughout learning. Selection of high learning rate may lead the algorithm to oscillate and become unstable, while selection of small learning rate will result in longer time taken for the algorithm to converge to the desired minimum point.

By applying adaptive learning, the learning rate is allowed to change during the training process. This algorithm will keep the learning step size as large as possible while keeping learning stable [6]. The learning rate is changed in such a way that it will be increased if stable learning is obtained per instance or decreased when the learning becomes unstable.

2.2.2 Quasi-Newton Method

Quasi-Newton method is a recommended technique for small sized networks (weights and inputs are less than hundred). Quasi-Newton is a batch update algorithm. As referred to [9], it works out the average gradient of the error surface across all cases before updating weights once at the end of an epoch. Since this is a batch update algorithm, it is unnecessary to select momentum or adaptive learning rates, which makes this method easy.

Generally, the updated variable is adjusted according to the following formula:

$$x = x + a.\ dX \qquad (2.2)$$

where x are weight / bias variables, dX is search direction and a is the selected line search algorithm. There are various line search algorithm that could be used with Quasi-Newton method, which includes Brent search, secant, golden section and backtracking search. For this research, backtracking search is set as default for the network training. In this search routine, the step multiplier is initialized at 1 and then it backtracks until an acceptable reduction in the performance is obtained.

The first search direction is the negative of the gradient performance while in the succeeding iterations, the search direction is obtained by the following formula:

$$dX = -H / gX \qquad (2.3)$$

where gX is the gradient and H is the approximated Hessian matrix.

CHAPTER 3

REVIEW ON CAR MODELS, ROAD PREVIEW MODELS AND OPTIMAL PREVIEW CONTROLLER

In previous work by [1], an optimal preview controller is implemented to follow simulated paths. Linear and nonlinear car models are designed to incorporate with the controller, as have been described by [2]. The first section of this chapter outlines the two car models, the next section describes the road preview models and the final section explains the optimal preview controller. Detailed explanation on the car models, road preview models and optimal controller could be retrieved from [2].

3.1 CAR MODELS

3.1.1 Linear Car Model

As illustrated by [1] and repeated by [2], the vehicle model is of standard yaw / sideslip type. It is assumed that the car is a rigid body, moves on flat paths with three degrees of freedom: forward, lateral (side) and yawing (side to side) motions. There are four types of forces of the vehicle model: front axle longitudinal force, front axle lateral force, rear axle longitudinal force and rear axle lateral force. Aerodynamic forces are discarded for this study, as they are considerably insignificant at normal speed for normal cars. The input to the car is the steering wheel angle.

In practice, speed should be reduced if the vehicle is nearing a curve or changing direction. However, for simplicity, the car moves only in forward direction with a constant speed throughout the whole path.

The parameters of the vehicle are as in the following Table 1:

Parameters	Values
Body Mass (M)	1200 kg
Yaw Inertia (I_z)	1500 kgm^2
Distance from center of gravity to front axle (a)	0.92
Distance from center of gravity to rear axle (b)	1.38
Cornering stiffness of front axle tyres (C_f)	1.2x10^5 Nrad^{-1}
Cornering stiffness of rear axle tyres (C_f)	8x10^4 Nrad^{-1}
Fixed Steering Ratio (Hand wheel / road wheel), G	17

Table 3.1: *Vehicle Model Parameters*

The state space equations of motion of the car model is $\dot{x}$ = Ax + Bδ_{sw} with the state vectors:

x = [x_1 x_2 x_3 x_4]^T where x_1 is global lateral position y,

x_2 is global lateral speed $\dot{y}$,

x_3 is global attitude angle Ψ,

x_4 is global attitude rate $\dot{\psi}$.

and

$$A=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -(C_f+C_r)/Mu & (C_f+C_r)/M & (bC_r-aC_f)/Mu \\ 0 & 0 & 0 & 1 \\ 0 & (bC_r-aC_f)/I_z u & (aC_f-bC_r)/I_z & -(a^2C_f+b^2C_r)/I_z u \end{bmatrix}, B=\begin{bmatrix} 0 \\ C_f/MG \\ 0 \\ aC_f/I_z G \end{bmatrix}.$$

The equations of motion are transformed to discrete time using the MATLAB command 'c2d'. Taking A_d and B_d as discrete matrices, the equation of motion becomes x(k+1) = A_dx(k) + $B_d\delta_{sw}$(k) in which k is the sampling time and T is the sampling interval. The sampling period is initially set as 0.05 s, and could be reduced when vehicle moves in higher speed to increase the number of preview points for the car controller. The preview points will be explained in the next section.

3.1.2 Non-Linear Car Model

The assumptions as well as the parameters of the non-linear car model are set to be similar as the previous linear car model. The difference between both models is the calculation of lateral tyre forces, which according to [2], are calculated using the Magic Formula by Bakker, Nyborg and Pacejka. All the Magic Formula parameters are considered constant and correspond to dry surface. The parameters of the formula are given as in Table 2 below:

Parameters	Values (N)
Stiffness, one tyre (b_m)	17.5
Shape, one tyre (c_m)	1.68
Peak, one front tyre (d_{mf})	3840
Peak, one rear tyre (d_{mr})	2560
Curvature (e_m)	0.6

Table 3.2: *Magic Formula Parameters*

In discrete state-space model, the non-linear car model is repeated from [2], given by the following forms:

$x_1(k+1) = x_1(k) + T.x_2(k)$

(3.1)

$x_2(k+1) = x_2(k) + T[\frac{1}{M}.(F_{yf}(k)+ F_{yf}(k))]$

$x_3(k+1) = x_3(k) + T.x_4(k)$

$x_4(k+1) = x_4(k) + T[\frac{1}{I_z}.(a.F_{yf}(k)- b.F_{yf}(k))]$

3.2 ROAD PREVIEW MODEL

Four paths are considered for the study: sinus path, lane change, sudden change of direction and smooth random path. By considering constant forward speed, the paths can be described by the lateral deviation, y_r, from a fixed straight line (x-axis) at sampling time kT.

In the global point of view, the road information is stored in the lateral deviations y_{rn} from a fixed x-axis at the time kT, corresponding to a specified forward speed u. Figure 3.1 shows the path errors in the global frame.

Taking n as the number of preview values, the lateral deviations at time kuT meters ahead of the car could be represented as $y_{ref}(k) = [y_{r0} \quad y_{r1} \quad \ldots \quad y_{rn}]^T$. The uT is the x spacing, in which u is the speed of the vehicle. Figure 3.1 shows the car and the road at instant k. At the next instant (k+1)T, the first road preview sample is discarded and the second sample of $y_{ref}(k)$ becomes the first value for $y_{ref}(k+1)$ and so on. For simplicity, the last sample value becomes the input to the system and the other n samples are regarded as states.

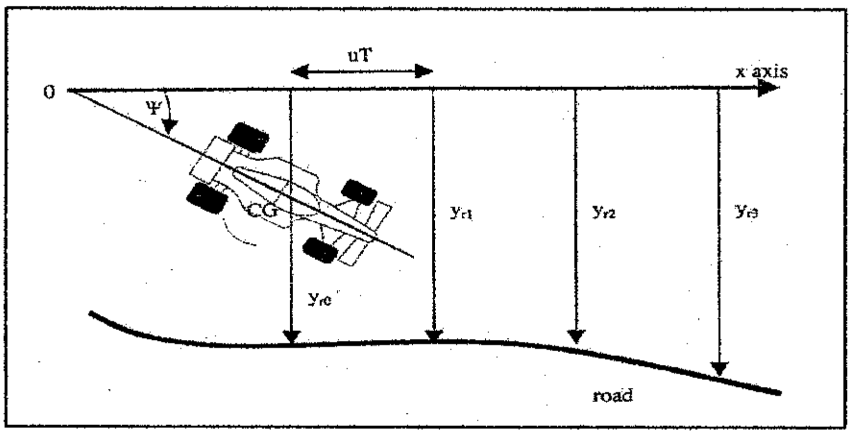


Figure 3.1: *Car and Road at instant k (adapted from [2])*

Taking y_{ref} as the state vector and y_{ri} as the input to the road system, the state space equation for the road preview model is $y_{ref}(k+1) = D.\ y_{ref}(k) + E.y_{ri}$. The vectors of D and E are:

$$D = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \quad \text{and } E = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{bmatrix}.$$

In the local point of view, the road information is stored in the lateral deviations y_{rh} from the local x-axis of the car, as depicted in Figure 3.2:

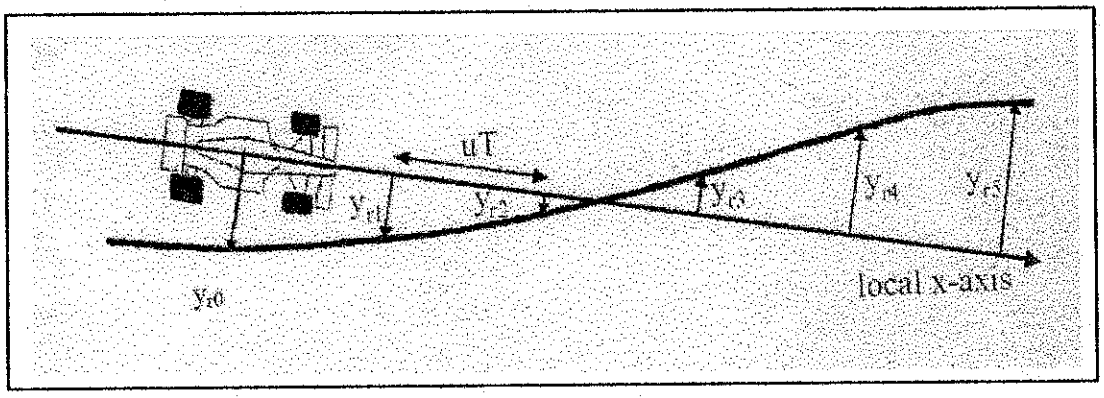


Figure 3.2: *Road Preview Model Local Point of View (from [1])*

3.3 OPTIMAL CONTROLLER

The purpose of the controller implementation is to establish a connection between the road preview model and the car. In other words, the car is to be driven along the path with the aid of the optimal controller. The state space equation of the car and the road (having no connection between both) is as follows:

$$\begin{bmatrix} x(k+1) \\ y_r(k+1) \end{bmatrix} = \begin{bmatrix} A_d & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} x(k) \\ y_r(k) \end{bmatrix} + \begin{bmatrix} 0 \\ E \end{bmatrix} y_{ri} + \begin{bmatrix} B_d \\ 0 \end{bmatrix} \delta_{sw}$$

(3.2)

The Linear Quadratic Gaussian (LQG) is then assessed with the following cost function:

$$J = \lim_{n\to\infty} \sum_{k=0}^{n} z^T(k).R_1.z(k) + \delta_{sw}(k).R_2.\delta_{sw}(k) \text{ with } Z = [X \ \ y_r]^T$$

(3.3)

in which $R_1 = C^T.Q.C$ with $C = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1/uT & -1/uT & 0 \ .. & 0 \end{bmatrix}$ and $Q = \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix}$ corresponding to the state vector $Z = [y \ \ \dot{y} \ \ \varphi \ \ \dot{\varphi} \ \ y_{r0} \ \ \ y_{r1} \ \ \ldots \ y_{rh}\ ]^T$, with $R_2 = 1$.

R_1 reflects the path following priorities, namely the path errors and the attitude angle errors while R_2 represents the importance attached to the control input. It is assumed that the pair (A, B) is stabilizable to guarantee existence, pair (A, $Q^{1/2}C$) is detectable for stability and R_2 to be positive definite to ensure finite control energy.

The works by Louam [10] and Prokop [11] show that the time-invariant optimal control, minimizing the cost function J is $\delta_{sw}(k) = -K.z(k)$. The vector gain K is determined by first solving the non-preview model, $x = [y \ \ \dot{y} \ \ \varphi \ \ \dot{\varphi}]^T$. Using the obtained result, the remainder of K which represents the preview control $y_r = [y_{r0} \ \ \ y_{r1} \ \ \ldots \ y_{rh}\ ]^T$ is solved.

Several controllers can be set up by changing the priorities in the cost function. If the priority is path following, q_1 is set to be 100 and q_2 is 0. If the priority is to keep the car tangential to the path, q_1 is set to be 0 and q_2 is 100. On the other hand, if priority is based on controlling the steer input and roughly following the path, q_1 is 0.05 and q_2 is 0. For all cases, R_2 is set to unity.

In this study, the priority is concentrated on path following. According to simulation results obtained by [1] and repeated by [2], as the speed of vehicle is increased, the preview gain will be more oscillatory. Figure 3.3 shows the simulation result for the optimal preview gains of path following.

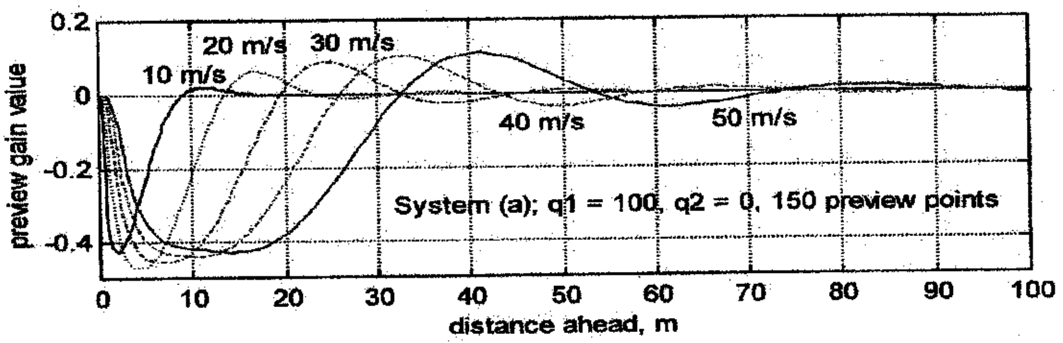


Figure 3.3: *Optimal Preview Gains for Path Following for Five Different Speeds (from [1])*

It should be noted that the controller is optimal as it is able to minimize the cost (3.3). However the optimal gain K is obtained due to the selection of matrices Q and R, which are the cost priorities. Without further adaptation, the matrices selection might not be the best selection. Therefore further modification to gain K can be implemented to obtain a better performance minimization. The final values of K may differ from the initial values of K. The gain update could be implemented using learning algorithms that will be highlighted in the next chapters.

CHAPTER 4

NEURAL NETWORK STEERING CONTROL OF A LINEAR CAR MODEL

Four paths (sinus shaped, lane change, sudden change of direction and smooth random path) were simulated and tracked with the use of an optimal controller. From previous works done by Sharp in [1], it is proven that the optimal controller has the capability to precisely track reasonable paths.

Dandré [2] has continued the research by tracking the similar paths using neural networks. The coefficients obtained through the optimal control theory were taken as the initial weighting parameters for the neural controller. The results proved that most of the time, neural controllers can perform significantly better than the conventional optimal controller.

This section is an upgraded version of works done by [2] for a linear car model. Previously, the network was trained using the online gradient method. Batch training (Gradient and Quasi-Newton) is now introduced for comparison. Updated weights, learning rates and time taken after final epochs are discussed.

4.1 IMPLEMENTATION USING GRADIENT METHOD

4.1.1 Neural Network Controller Implementation

The controller is set to be a linear, single processing neuron. The input to the controller is the augmented state $z = [x \quad y_r]^T$. x is obtained from the equations of motion of the car model while y_r is the local lateral preview errors. The output of the neuron is the steering wheel angle, δ_{sw} which was represented by [2], in the following formula:

$$\delta_{sw} = w_1(k).z_1(k)+w_2(k).z_2(k)+\ldots.w_{n+5}(k).z_{n+5}(k) \qquad (4.1)$$

By considering n preview points, there would be 4+n+1 weighting parameters and one bias for the single neuron. The weighting parameters are set in such way as there are four non-preview system (states x) and n+1 preview points at instant k. As it is desired that all path following errors be minimized, the best steering wheel angle would be zero when the car is moving on a straight path. Thus the bias b is set to zero.

Using linear quadratic cost function, the vehicle performance is evaluated according to formula (3.3). From the equation, the partial derivatives of the cost with respect to the augmented state $(\partial J(z(k),\delta_{sw}(k)/\partial z(k))$ and the partial derivatives of the cost with respect to control variable $(\partial J(z(k),\delta_{sw}(k))/\delta_{sw}(k))$ can be obtained. As the car is supposed to follow the simulated paths, the cost priorities are set as:

$$q_1=100,\ q_2 = 1,\ R_2 = 1 \text{ and } R1=\begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix}.$$

As was done in previous works, the initial weighting parameters W_0 for the neural controllers were taken from coefficients obtained from the optimal control theory. Alternatively the initial weighting parameters could also be set either to zero, or chosen randomly. However, it is preferred to take the obtained coefficients from the optimal control theory as it gives the best representation of the path tracking optimization.

A high learning rate may lead to instability of the algorithm whilst a low learning rate may cause longer time for the algorithm to converge to desired performance. By running the simulation for a number of times, the best initial learning rates were chosen based on the least maximum y-path error obtained after the simulation. To ensure an improved performance of the steepest gradient descent algorithm, the learning rate is allowed to be adaptive, i.e. it is allowed to change during the training process. By using [4], the learning rate is multiplied by 1.05 if the cost ratio between the present cost and previous cost is less than 1. On the other hand, it is multiplied by 0.7 if the cost ratio is more than 1.005.

4.1.2 Simulation by Online Training

In works by Dandré [2], the network was trained for one epoch. One epoch is equivalent to one whole simulated path length minus the number of preview points. The preview points are arbitrarily set to 40 for all cases. For some paths, network training for one epoch would be sufficient, but in some cases, by training for several epochs, the network performance would be improved, which in turn reduces the maximum y-path error. The behaviour of the learning rates and the updated weights per epochs could also be observed. For this section, the number of training epochs is set to five.

A. Sinus Path (at 20m/s, 40 preview points)

The path following is as shown in Figure 4.1(i). Initially, after the first epoch, the maximum steady-state path error is $6.5\text{x}10^{-4}$ m (Figure 4.1(ii)). At the first epoch, the learning rate and the updated weights oscillate a little and significantly reduced to some steady-state values (Figures 4.1(iii, iv)). By training the network up to five epochs, the maximum steady-state path error is reduced to $2\text{x}10^{-4}$ m (Figures 4.1 (v)). The path errors during the first and final epochs are significantly less that the errors generated by the optimal controller. The learning rates become very small while the updated weights settle to some constant values after certain epochs (Figures 4.1(vi, vii)). The maximum steering wheel angle is 0.2 radians (Figure 4.1 (viii). The observations for the sinus path are summarized in the following Table 4.1:

1.Sinus Path	
Initial Learning Rate	0.1
Path Distance	900m
Maximum y-path Error (First Epoch)	6.5×10^{-4} m
Maximum y-path Error (Final Epoch)	2×10^{-4} m
Final Learning Rate (First Epoch)	1.1113e-017
Final Learning Rate (Final Epoch)	1.8971e-256
Learning Time (s) – First Epoch	10.215
Learning Time (s) – Final Epoch	48.388
Final Weight (at 10th Point)	-0.9813

Table 4.1: *Summarized Observations for Sinus Path Following*

B. Lane Change (at 20m/s, 40 preview points)

The path following is as shown in Figure 4.2(i). The maximum y-path errors for the first and final epochs are similar, at 8×10^{-3} m (Figure 4.2(ii)). Similar to the sinus path, the learning rate increases and the updated weights oscillate a little before reducing tremendously to steady-state values during the first epoch, as shown in Figures 4.2(iii, iv). After the first epoch, the learning rate continues to decrease while the updated weights vary insignificantly (Figures 4.2(v, vi)). The maximum steering wheel angle is shown in Figure 4.2 (vii). Table 4.2 summarizes the whole observations:

2. Lane Change	
Initial Learning Rate	0.05
Path Distance	300m
Maximum y-path Error (both cases)	$8x10^{-3}$ m
Final Learning Rate (First Epoch)	1.3608e-016
Final Learning Rate (Final Epoch)	7.4652e-075
Learning Time (s) – First Epoch	4.637
Learning Time (s) – Final Epoch	16.745
Final Weight (at 10th Point)	-0.9813

Table 4.2: *Summarized Observations for Lane Change Path Following*

C. Sudden Change of Direction (at 20m/s, 40 preview points)

The path following is as shown in Figure 4.3(i). The maximum y-path errors are similar during the first and final epochs (Figure 4.3 (ii)). The neural network controller has a slightly better performance than the optimal controller, judging by the obtained path errors. The behaviour of the learning rates and the updated weights are also parallel to the behaviours observed from the previous path following (Figures 4.3 (iii – vi)). The summary of the observation is as shown in Table 4.3:

3. Sudden Change of Direction	
Initial Learning Rate	0.3
Path Distance	200m
Maximum y-path Error (both cases)	0.065 m
Final Learning Rate (First Epoch)	6.7641e-008
Final Learning Rate (Final Epoch)	$7.2805x10^{-34}$
Learning Time (s) – First Epoch	3.465
Learning Time (s) – Final Epoch	9.955
Final Weight (at 10th Point)	-0.9813

Table 4.3: *Summarized Observations for Sudden Change of Direction*

D. Random Path (at 20m/s, 40 preview points)

The path following is as shown in Figure 4.4(i). The maximum y-path error after the first epoch reduces from $3x10^{-3}$ m to $2.48x10^{-3}$ m after the fifth epoch as shown in Figures 4.4 (ii, iii). The behaviour of the learning rates and the updated weights is also parallel to the behaviour observed from the previous path following (Figures 4.4 (iv – v)). The summary of the observation is as shown in Table 4.4:

4. Random Path	
Initial Learning Rate	0.1
Path Distance	900m
Maximum y-path Error (First Epoch)	$3x10^{-3}$ m
Maximum y-path Error (Final Epoch)	$2.48x10^{-3}$ m
Final Learning Rate (First Epoch)	$1.8633x10^{-52}$
Final Learning Rate (Final Epoch)	$5.8885x10^{-286}$
Learning Time (s) – First Epoch	8.393
Learning Time (s) – Final Epoch	40.344
Final Weight (at 10th Point)	-0.9813

Table 4.4: *Summarized Observations for Random Path*

Figures 4.1: Sinus Path at 20m/s, 40 preview points

(Solid: Neural Network, Dashed: Optimal Controller)

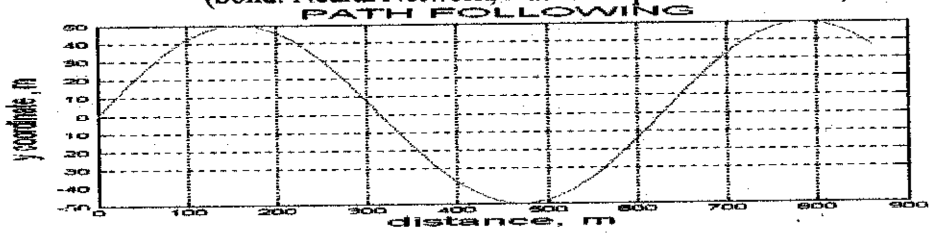


Figure 4.1(i): *Path Following (follows up until K-n-1)*

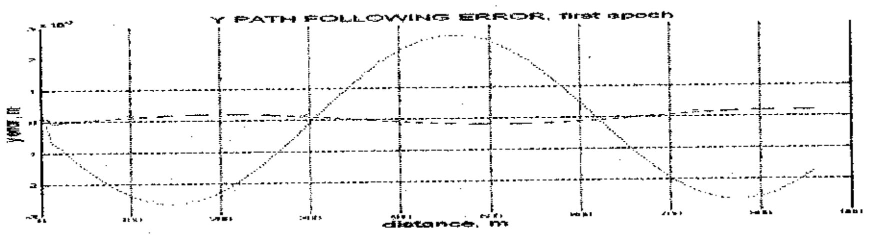


Figure 4.1(ii): *Maximum y-path error at first epoch (blue: neural controller, green: optimal controller)*

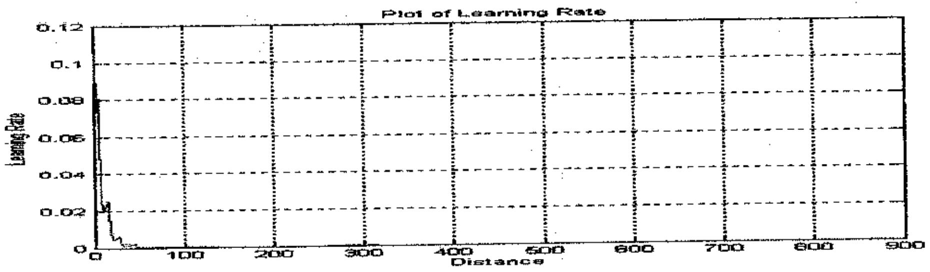


Figure 4.1(iii): *Plot of Learning Rate at First Epoch*

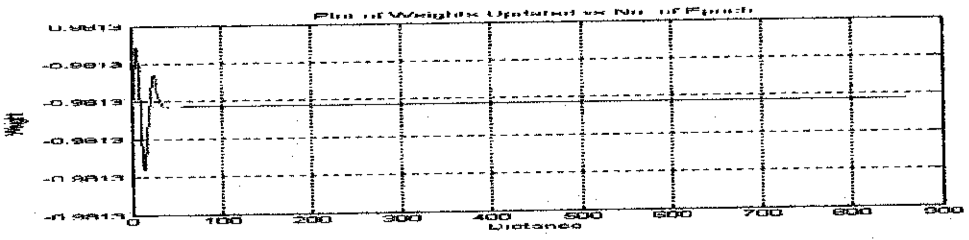


Figure 4.1(iv): *Plot of Updated Weight at First Epoch*

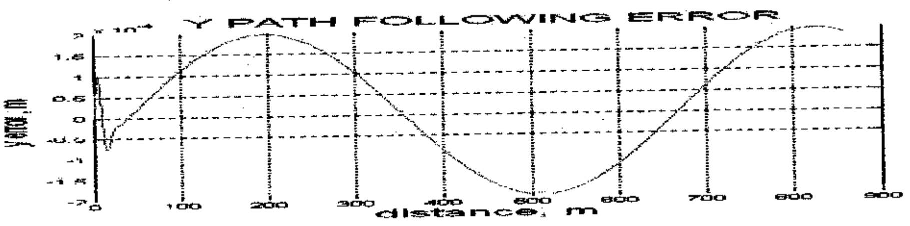


Figure 4.1(v): *Maximum y-path error at fifth epoch*

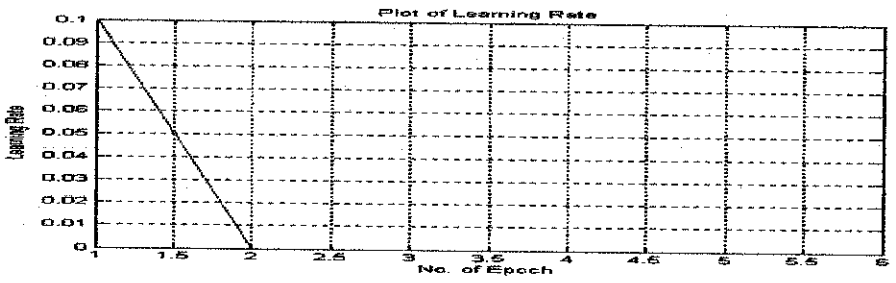


Figure 4.1(vi): *Plot of Learning Rate vs. No. Of Epoch*

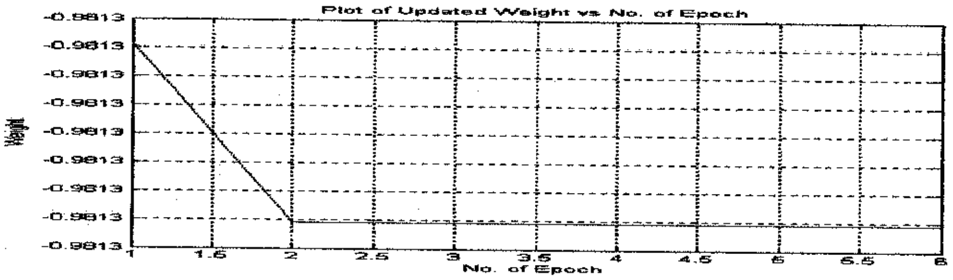


Figure 4.1(vii): *Plot of Updated Weight vs. No. Of Epoch*

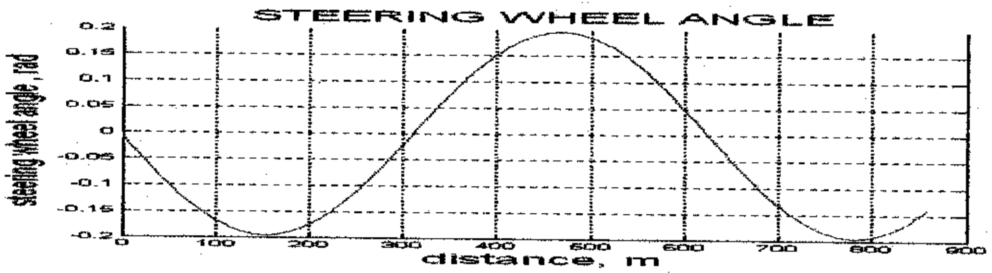


Figure 4.1(viii): *Network Output*

Figures 4.2: Lane Change at 20m/s, 40 preview points

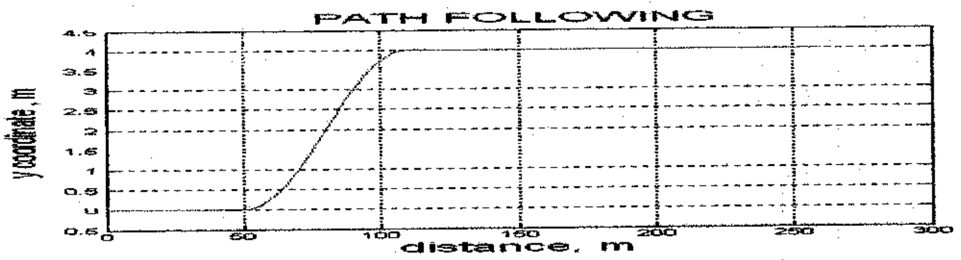


Figure 4.2(i): *Path following (follows up until K-n-1)*

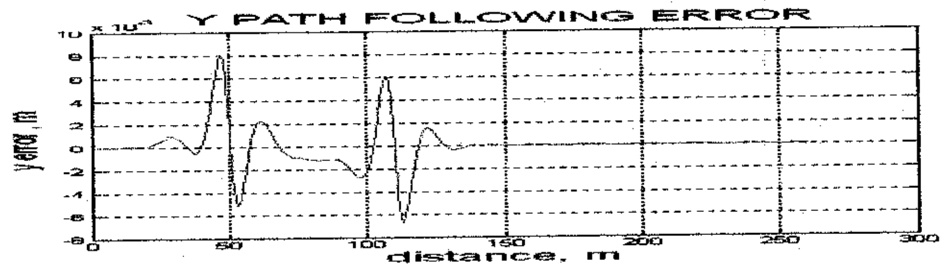


Figure 4.2(ii): *Maximum y-path error*

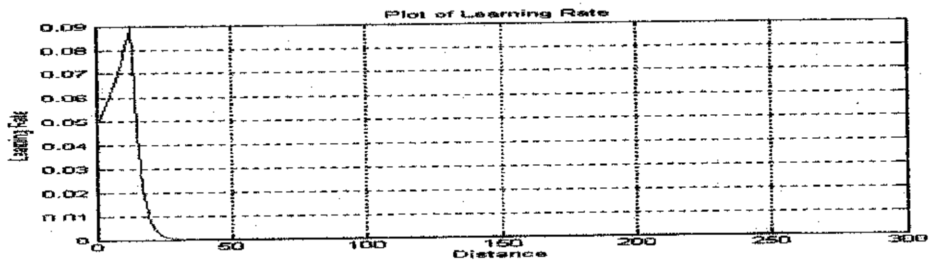


Figure 4.2(iii): *Plot of Learning Rate at First Epoch*

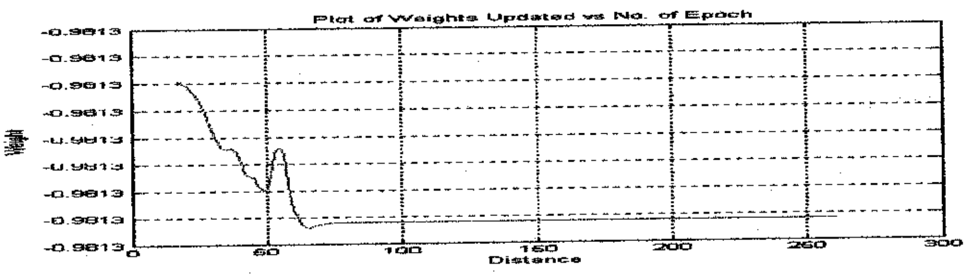


Figure 4.2(iv): *Plot of Updated Weight at First Epoch*

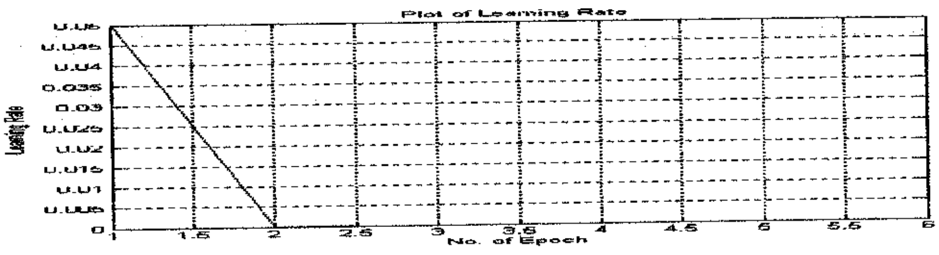


Figure 4.2(v): *Plot of Learning Rate vs. No. Of Epoch*

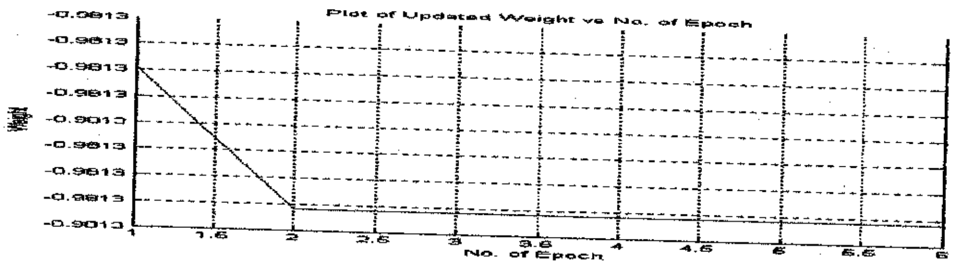


Figure 4.2(vi): *Plot of Updated Weight vs. No. Of Epoch*

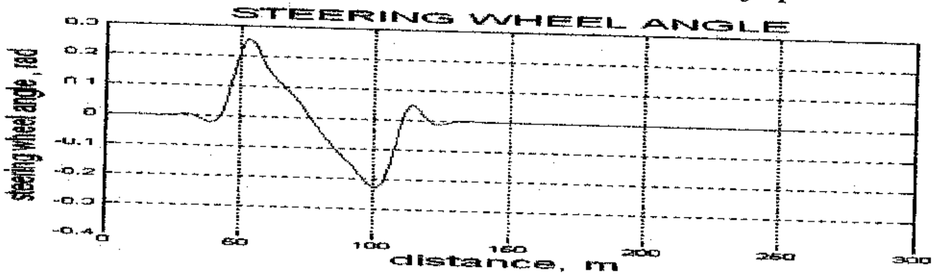


Figure 4.2(vii): *Network Output*

Figures 4.3: Sudden Change of Direction at 20m/s, 40 preview points

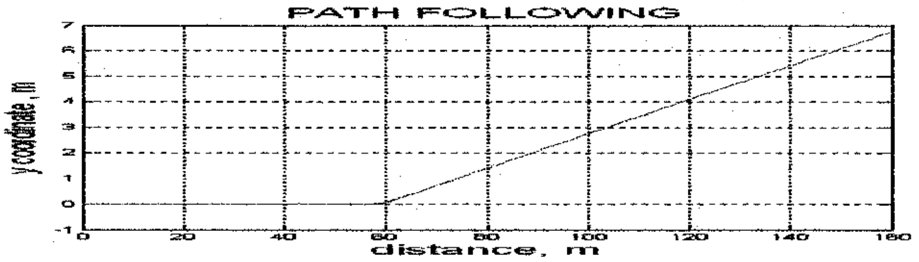


Figure 4.3(i): *Path following (follows up until K-n-1)*

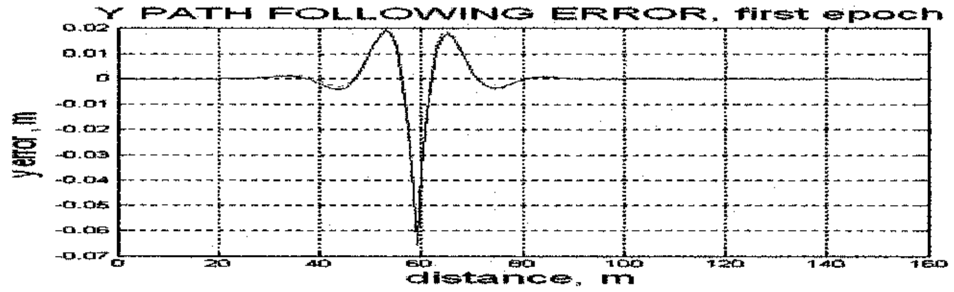


Figure 4.3(ii): *Maximum y-path error at first epoch*

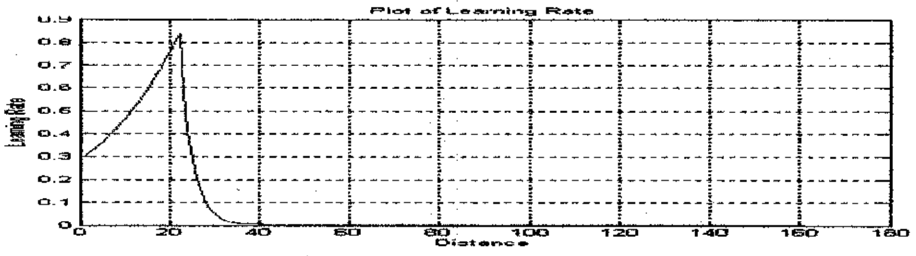


Figure 4.3(iii): *Plot of Learning Rate at First Epoch*

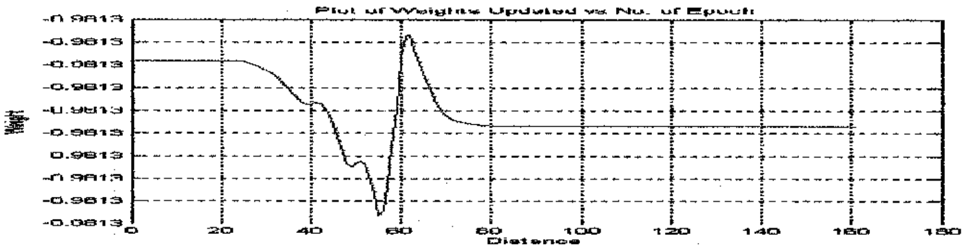


Figure 4.3(iv): *Plot of Updated Weight at First Epoch*

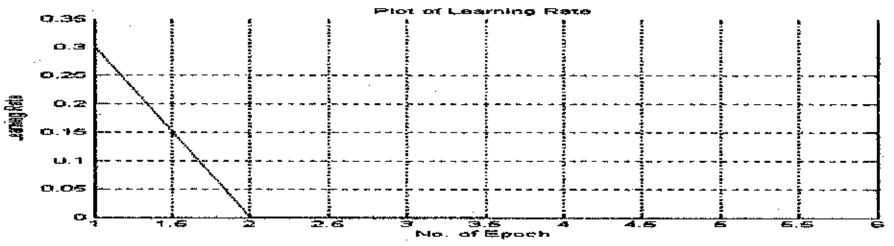


Figure 4.3(v): *Plot of Learning Rate vs. No. Of Epoch*

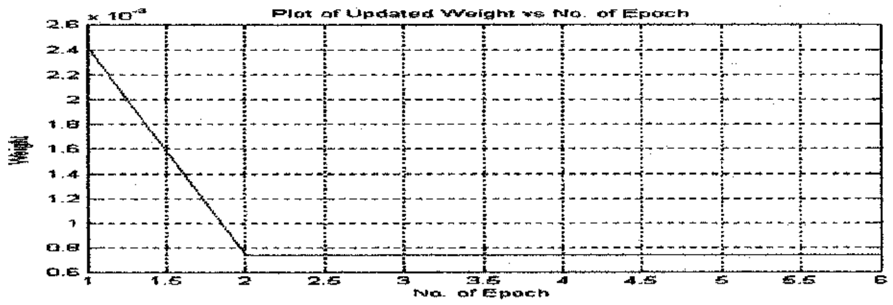


Figure 4.3(vi): *Plot of Updated Weight vs. No. Of Epoch*

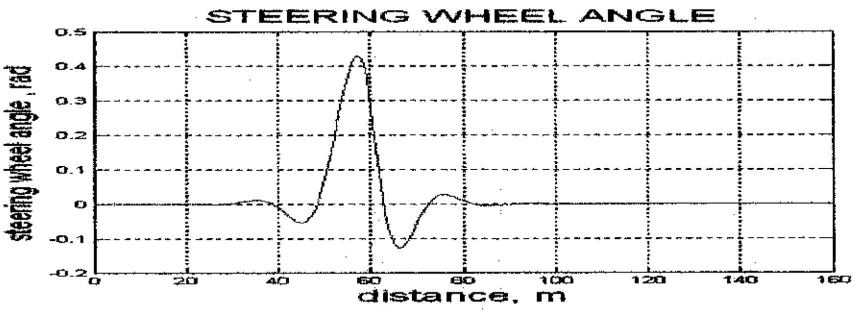


Figure 4.3(vii): *Network Output*

Figures 4.4: Random Path at 20m/s, 40 preview points

(Solid: Neural Network, Dashed: Optimal Controller)

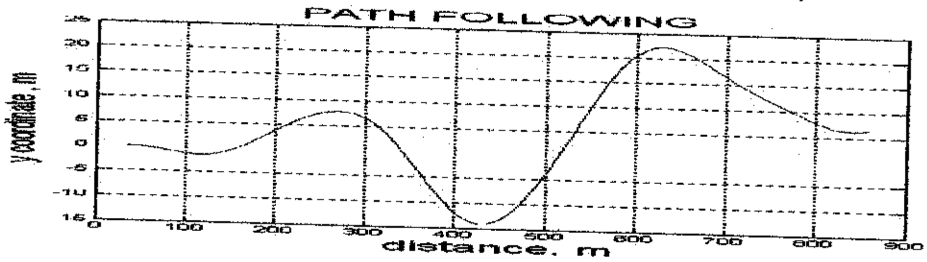


Figure 4.4(i): *Path following (follows up until K-n-1)*

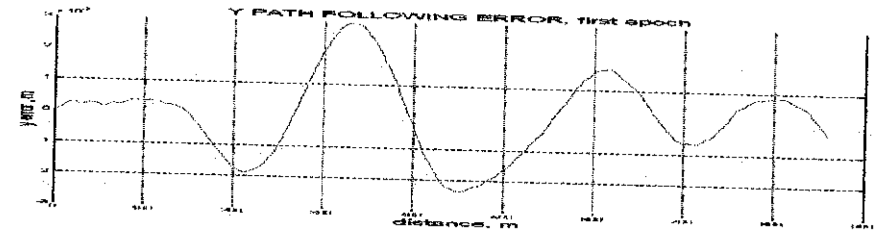


Figure 4.4(ii): *Maximum y-path error at first epoch*

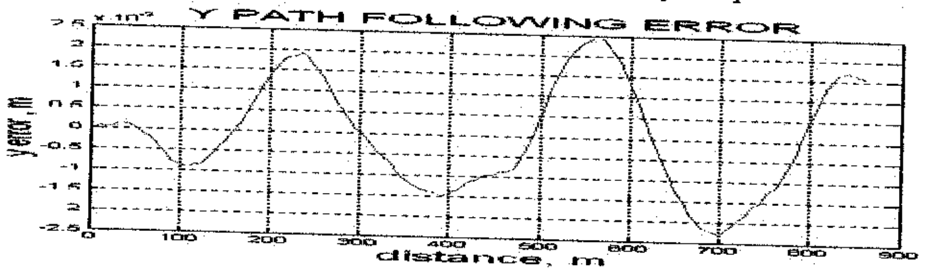


Figure 4.4(iii): *Maximum y-path error at fifth epoch*

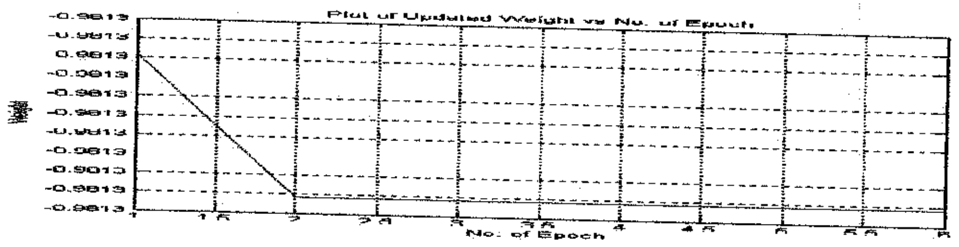


Figure 4.4(iv): *Plot of Updated Weight vs. No. Of Epoch*

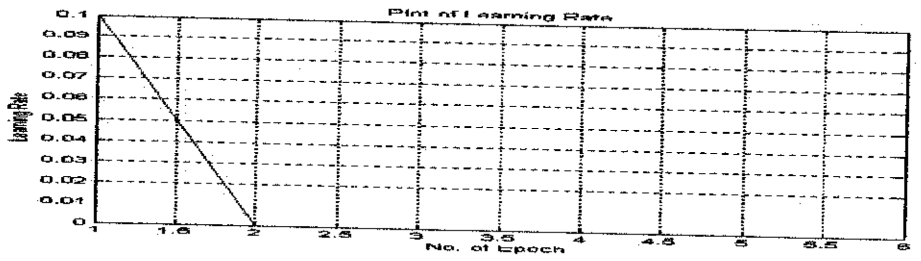


Figure 4.4(v): *Plot of Learning Rate vs. No. Of Epoch*

4.1.3: Simulation by Batch Training

The gradient method with adaptive learning rate could also be used for training in batches. The batch size is set to be the total distance of path minus the number of preview points, while the adaptive rate is set as default using MATLAB command 'traingda'. The network training will stop either when the maximum number of epochs is reached or the performance has reached the goal. For this training, the epochs are set to 10 while the performance goal is set to 1×10^{-10}. As the network deals with a linear car model, the transfer function that calculates the layer's output from its input is set as 'purelin'.

A. Sinus Path (at 20m/s, 40 preview points)

Using the similar path (as in Figure 4.1(i)), the maximum y-path error increases to 3×10^{-3} m with batch training (Figure 4.5(i)). The neural network controller has a slightly better performance than the optimal controller judging from the obtained maximum y-path error. It takes four epochs to converge to the performance goal (Figure 4.5(ii)). The maximum steering wheel angle remains at 0.2 m/s (Figure 4.5 (iii)). The training time is however shorter with batch learning as compared to the online learning.

B. Lane Change and Sudden Change of Direction (at 20m/s, 40 preview points)

The maximum y-path errors are reduced to 0.0075 m and 0.061 m for lane change and sudden change of direction respectively. For both paths, the learning takes less than one epoch to achieve the performance goal. The maximum steering wheel angles are similar between the batch and online training. The times taken for training are also shorter. Figure 4.6 and Figure 4.7 illustrate the simulated lane change and sudden change of direction.

C. Random Path (at 20m/s, 40 preview points)

The path following is as shown in Figure 4.8(i). The maximum y-path error is $2.3\text{x}10^{-3}$m (Figure 4.8(ii)). The performance goal at $1\text{x}10^{-10}$ is unachievable even after more than ten epochs. The maximum steering wheel angle is 0.18 radians, as shown in Figure 4.8 (iii).

Figures 4.5: Sinus Path at 20m/s, 40 preview points

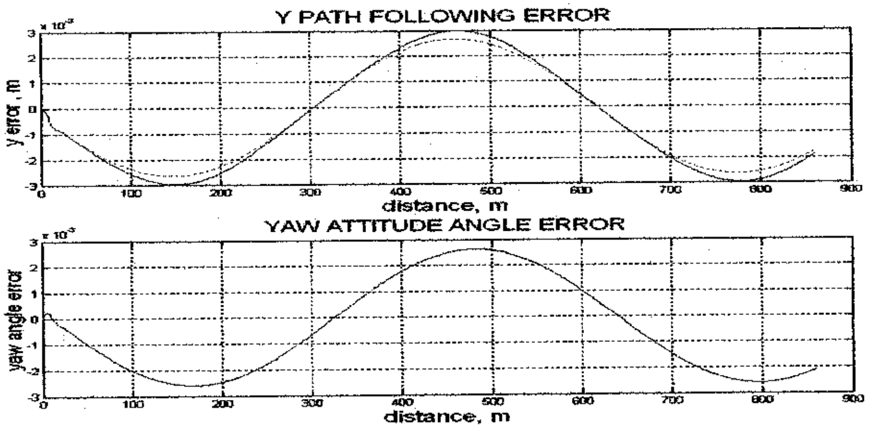


Figure 4.5(i) - (top): *y-path error*
Figure 4.5(i) - (bottom): *Yaw Attitude Angle Error*

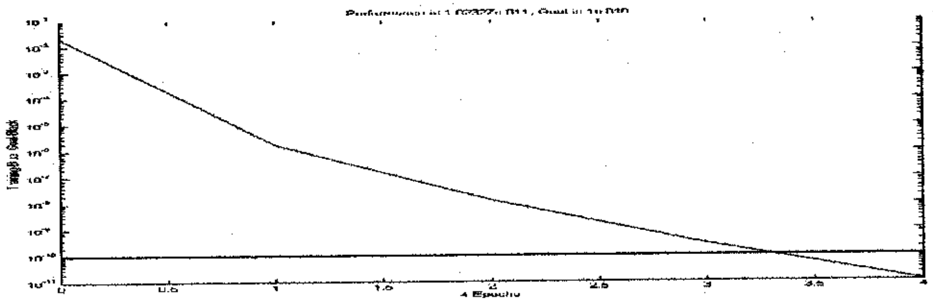


Figure 4.5(ii): *Training Result*

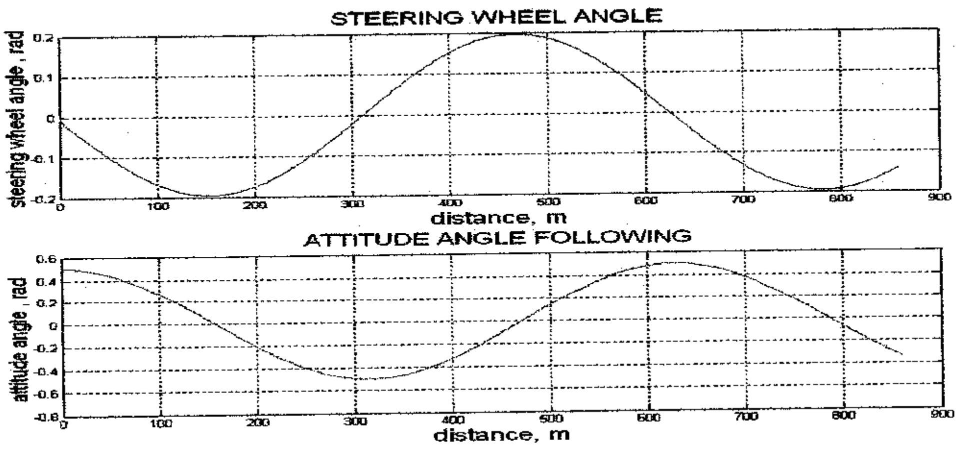
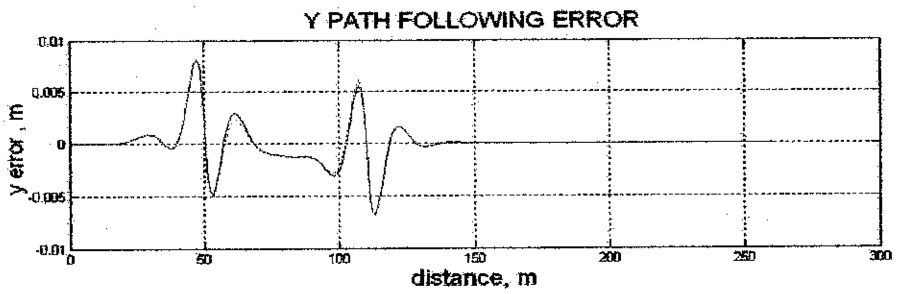


Figure 4.5(iii) - (top): *Steering Wheel Angle (Network Output)*
Figure 4.5(iii) - (bottom): *Attitude Angle Following*

Figures 4.6: Lane Change at 20m/s, 40 preview points



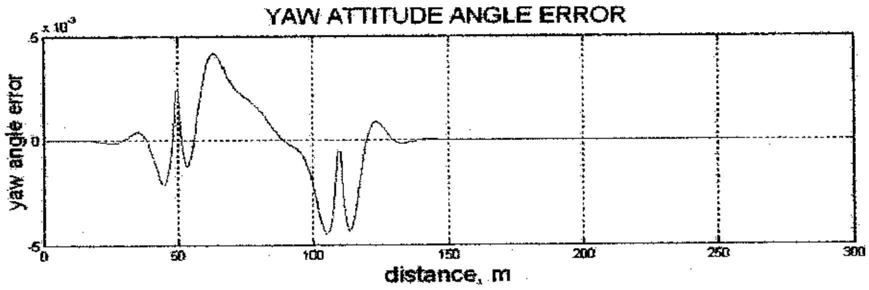
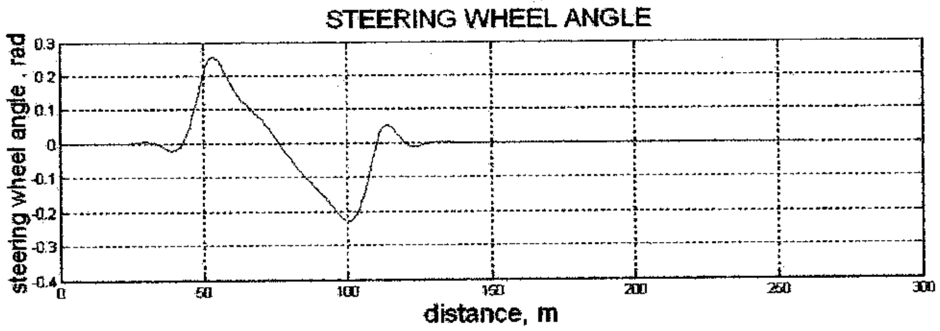


Figure 4.6(i) - (top): *y-path error*
Figure 4.6(i) - (bottom): *Yaw Attitude Angle Error*



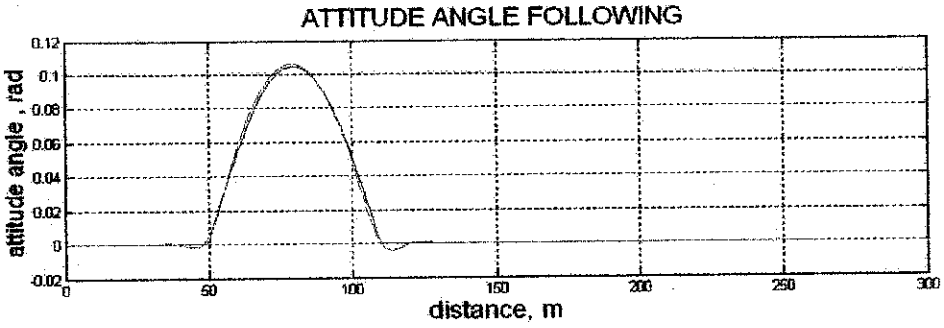
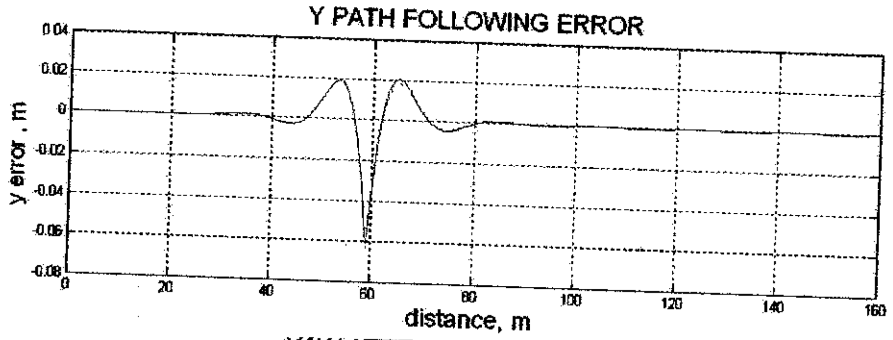


Figure 4.6(ii) - (top): *Steering Wheel Angle (Network Output)*
Figure 4.6(ii) - (bottom): *Attitude Angle Following*

Figures 4.7: Sudden Change of Direction at 20m/s, 40 preview point



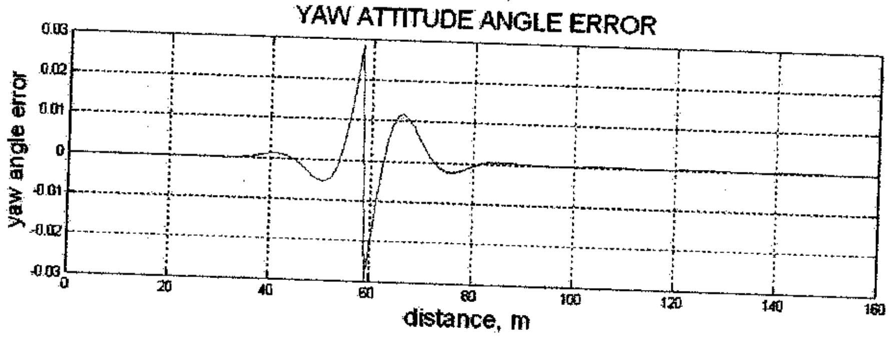
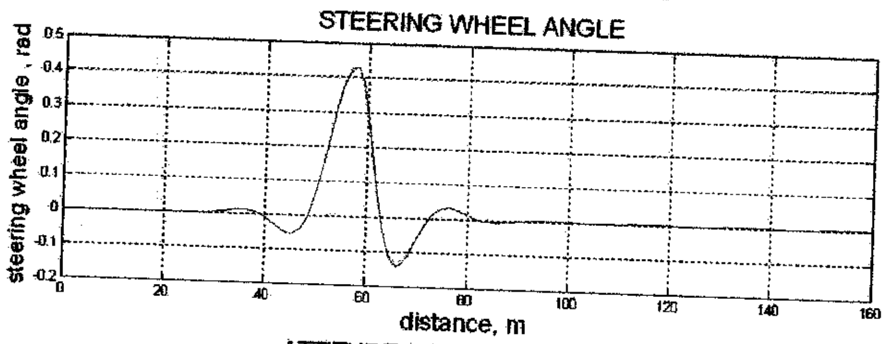


Figure 4.7 (i) - (top): *y-path error*
Figure 4.7 (i) - (bottom): *Yaw Attitude Angle Error*



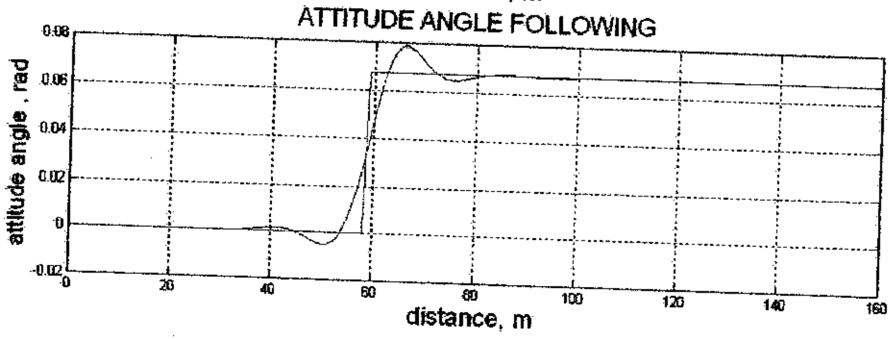


Figure 4.7 (ii) - (top): *Steering Wheel Angle (Network Output)*
Figure 4.7 (ii) - (bottom): *Attitude Angle Following*

Figures 4.8: Random Path at 20m/s, 40 preview point

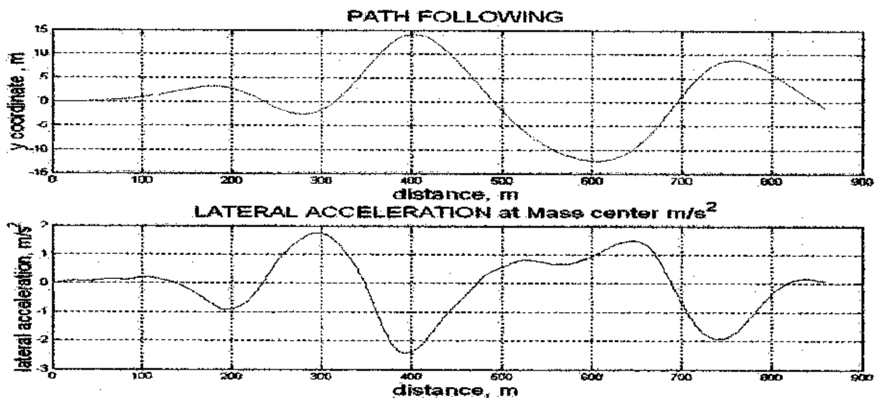


Figure 4.8(i) - (top): *Path Following (follows up until K-n-1)*
Figure 4.8(i) - (bottom): *Lateral Acceleration at final epoch*

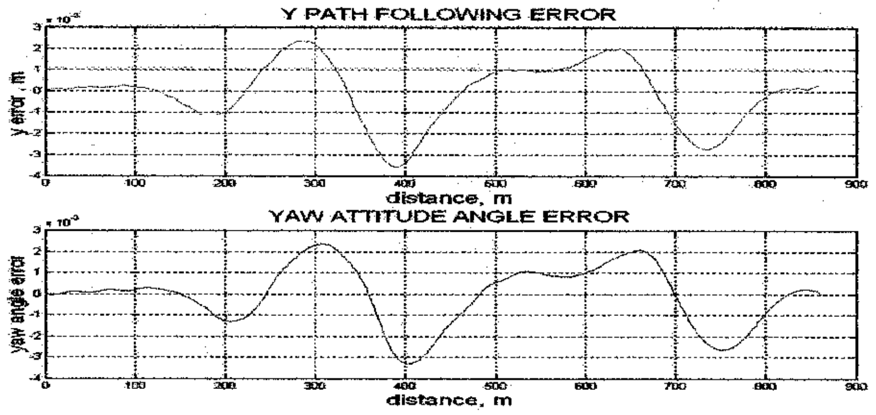


Figure 4.8(ii) - (top): *y-path error*
Figure 4.8(ii) (bottom): *Yaw Attitude Angle Error*

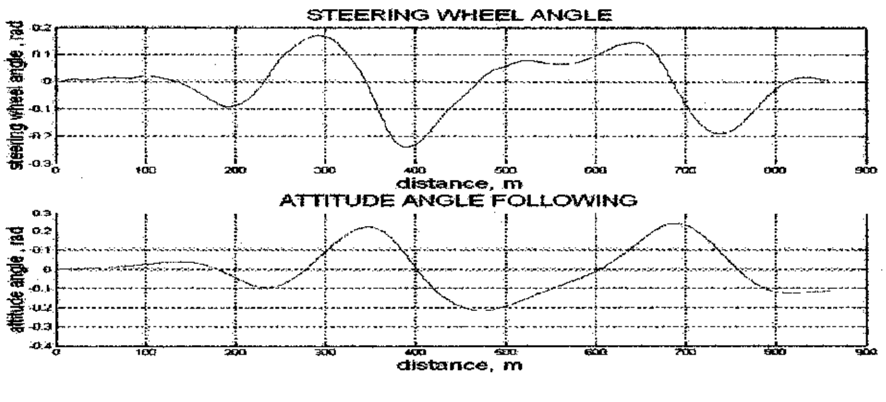


Figure 4.8(iv) - (top): *Steering Wheel Angle (Network Output)*
Figure 4.8(iv) - (bottom): *Attitude Angle Following*

4.2 IMPLEMENTATION USING QUASI-NEWTON METHOD

4.2.1 Neural Network Controller Implementation

The neural controller is set similarly as described in the previous section. The algorithm for network training differs in such a way that Quasi-Newton method can only be trained by batch [6]. According to [7], for a small-scaled network, Quasi-Newton method would be a good algorithm to use.

One batch is equivalent to one epoch, which is set to be the path distance minus the preview points. The MATLAB command 'trainbfg' is applied to the original coding. The one-dimensional minimization using backtracking method is set as the search routine default. For all four paths, the initial learning rates, initial weights, number of preview points, speed and path distances are similar to the previous cases. Similar to the previous batch training, the transfer function used for the network is 'purelin'.

4.2.2 Simulation Results

A. Sinus Path (at 20m/s, 40 preview points)

The maximum y-path error is 3×10^{-3}m, which is exactly equivalent to the maximum error obtained from gradient (batch) method (Figure 4.9(i)). However, the optimal controller has a slightly better performance than the neural controller. The maximum steering wheel angle is also 0.2 radians as obtained previously (Figure 4.9(ii)). However, it takes one epoch less with Quasi-Newton method as compared to the gradient method for the controller performance to converge to the specified goal (Figure 4.9(iii)). Table 4.5 summarizes the observations.

1. Sinus Path		
POINTS OF COMPARISON	**GRADIENT**	**QUASI-NEWTON**
Maximum y-path Error	$3\text{x}10^{-3}$	$3\text{x}10^{-3}$ m
Performance	1.02327e-011	1.20082e-011
Epoch to reach target	4	3
Learning Time (s)	8.051	8.011
Weight after last epoch (at 10th point)	-0.9812	-0.9812

Table 4.5: *Comparisons for Sinus Path between gradient and Quasi-Newton*

B. Lane Change (at 20m/s, 40 preview points)

The maximum y-path error is 0.0075 m, which is exactly similar to the observation of the gradient method (batch training), and slightly smaller than the result obtained through the online training. In comparison to the optimal controller, the neural controller has a slightly better performance. Both batch simulations only take less than one epoch to converge to the performance goal. The training time for the Quasi-Newton is however slightly greater than the gradient method (batch training), but absolutely less than for the online training. The final weight updated for both batch simulations are almost the same, which results in similar network output, which is the steering wheel angle history. The summarized result is in Table 4.6.

2. Lane Change		
POINTS OF COMPARISON	**GRADIENT**	**QUASI-NEWTON**
Maximum y-path Error	0.0075 m	0.0075 m
Performance	$1.64141\text{x}10^{-13}$	$1.64141\text{x}10^{-13}$
Epoch to reach target	Less than 1	Less than 1
Learning Time (s)	2.804	2.894
Weight after last epoch (at 10th point)	-0.9813	-0.9813

Table 4.6: Comparisons *for Lane Change between gradient and Quasi-Newton*

C. Sudden Change of Direction (at 20m/s, 40 preview points)

The maximum y-path error is similar for both batch training, and slightly less than the previous online training. It takes less than one epoch for both batch simulations to converge to the performance goal. The time taken with Quasi-Newton method is however greater than the previous gradient (batch) method, but less than the online training time. The final weights obtained for both batch simulations are almost the same, which results in almost similar network output. The summary is in Table 4.7 below:

3. Sudden Change of Direction		
POINTS OF COMPARISON	**GRADIENT**	**QUASI-NEWTON**
Maximum y-path Error	0.061 m	0.061 m
Performance	4.78561e-017	4.78561e-017
Epoch to reach target	Less than one epoch	Less than one epoch
Learning Time (s)	2.213	3.335
Weight after last epoch (at 10th point)	-0.9813	-0.9813

Table 4.7: Comparisons *for Sudden Change of Direction between gradient and Quasi-Newton*

D. Random Path (at 20m/s, 40 preview points)

The observations between both batch training methods are similar except that it takes less training time for the Quasi-Newton method as compared to the gradient method (both batch and online training). Another significant behaviour is that by Quasi-Newton, the performance goal could be achieved within only one epoch, but was unachievable with the gradient method (Figure 4.10). The neural controller has a slightly better performance than the optimal controller. The observation summary is as shown in Table 4.8.

4. Random Path		
POINTS OF COMPARISON	**GRADIENT**	**QUASI-NEWTON**
Maximum y-path Error	$2.3\text{x}10^{-3}$	$3\text{x}10^{-3}$
Performance	Performance goal is not achieved	$1.7641\text{x}10^{-13}$
Epoch to reach target	-	Less than one epoch
Learning Time (s)	8.242	8.102
Weight after last epoch (at 10th point)	-0.9813	-0.9813

Table 4.8: Comparisons *for Random Path between gradient and Quasi-Newton*

Figure 4.9: Sinus Path at 20m/s, 40 preview point

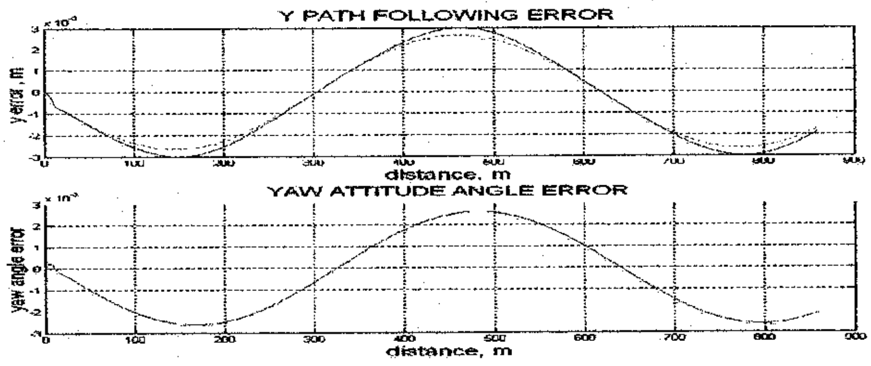


Figure 4.9(i) - (top): *y-path error (dashed: optimal controller, solid: neural controller)*
Figure 4.9(ii) - (bottom): *Yaw Attitude Angle Error*

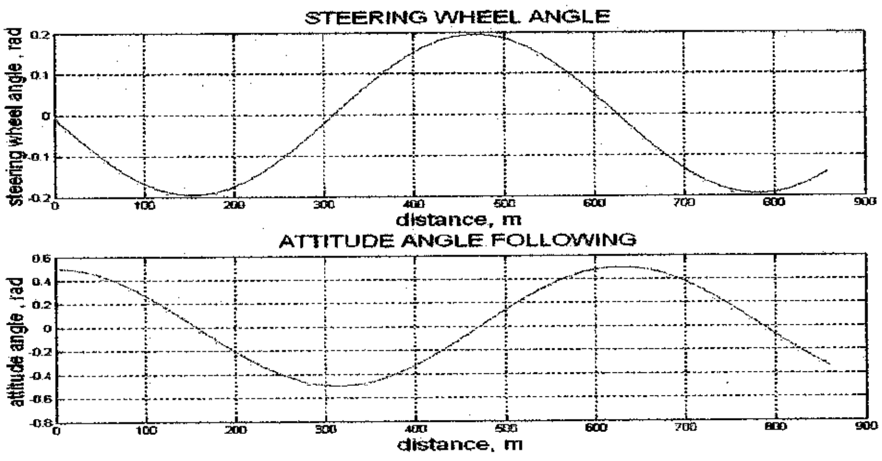


Figure 4.9 (ii) - (top): *Steering Wheel Angle (Network Output)*
Figure 4.9 (ii) - (bottom): *Attitude Angle Following*

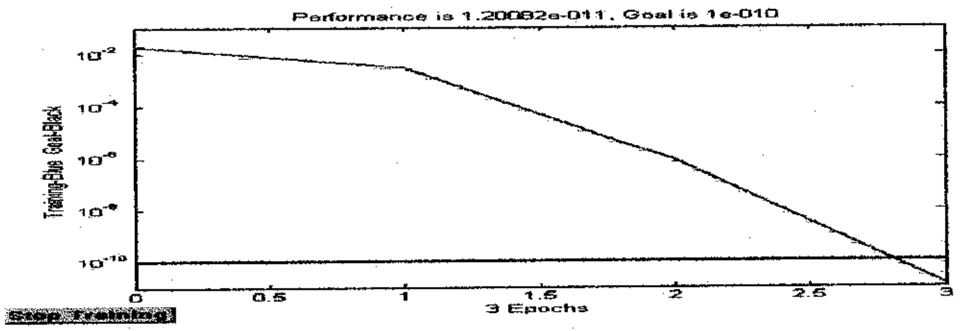


Figure 4.9 (iii): *Training Result*

Figures 4.10: Random Path at 20m/s, 40 preview point

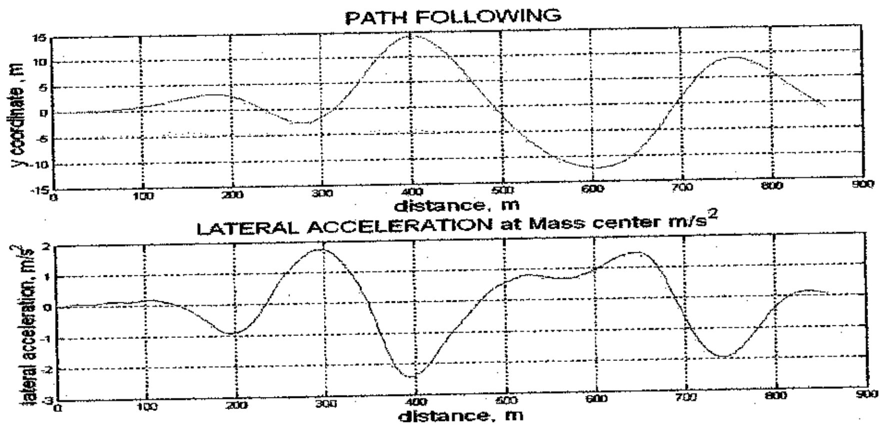


Figure 4.10(i) - (top): *Path Following (follows up until K-n-1)*
Figure 4.10(i) - (bottom): *Lateral Acceleration at final epoch*

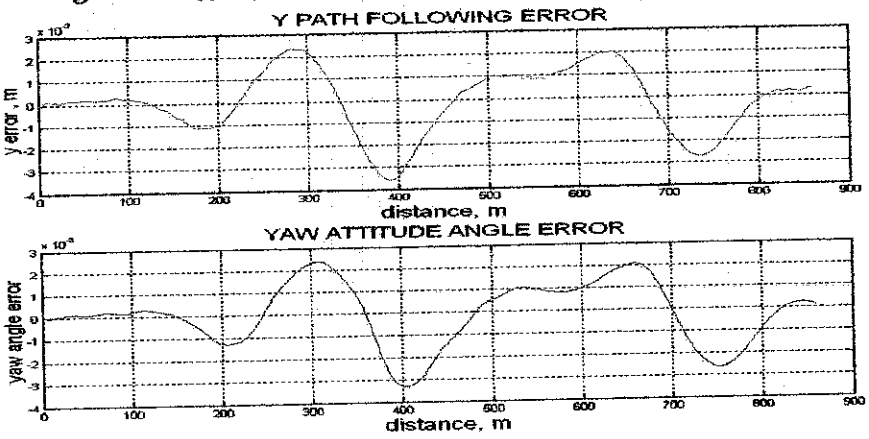


Figure 4.10(ii) - (top): *y-path error*
Figure 4.10(ii) (bottom): *Yaw Attitude Angle Error*

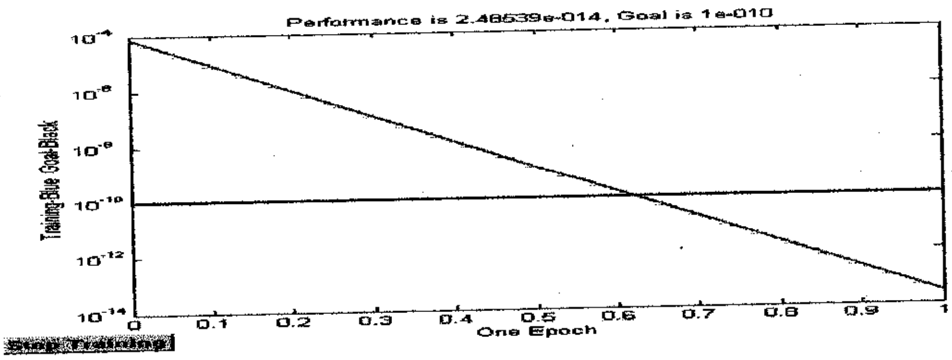


Figure 4.10 (iii): *Training Result*

4.3 PERCENTAGE OF AVERAGE WEIGHT CHANGE

For all cases, the percentages of average weight change were calculated to see how much the weights were updated at the last epoch. The formula used is as follows:

$$\text{Average Weight Change} = \frac{\sum_{i=1}^{n} \left| \left(\frac{W_{NN(i)} - W_{OC(i)}}{W_{OC(i)}} \right) \right| x100\%}{n} \qquad (4.2)$$

where W_{NN} is the updated weight by neural network controller

W_{OC} is the original weight of the optimal controller, and

n is total number of weights

The summarized calculation of the percentage is as shown in the following Table 4.9:

	Quasi-Newton (%)	Gradient (Batch) (%)	Gradient (Online) (%)
1) Sinus	46.7337	69.4258	7.6762
2) Lane Change	2.3245	2.3245	3.4057
3) Sudden Change	2.3245	2.3245	7.4521
4) Random Path	12.262	12.2544	3.6942

Table 4.9: *Linear Car Model: Percentage of Average Weight Change*

4.4 DISCUSSIONS

Through online training with gradient method, the final weight oscillates before decreasing rapidly during the first epoch and later settles to some steady values in the subsequent epochs. This is in parallel with the behaviour of the learning rates. As the trial progresses, the learning rates will either jump up or oscillate, depending on the type of path, before decreasing rapidly in the first epoch. For the next epochs, the rate decreases slowly, which results in insignificant changes to the updated weights. This in turn led to similar network output (the steering wheel angle) for the successive epochs. However, through few iterations, the path following could be improved, as proven through the sinus and random paths.

For batch training, although the path following error and the maximum angle of the steering wheel are similar for gradient and Quasi-Newton methods, the training time and the ability to converge to the performance goal makes the latter superior to the former. This proves the theory that although Quasi-Newton requires more computation in each iteration, it usually converges in fewer iterations.

The percentage of average weight change could be compared between the three modes of training (gradient-online, gradient-batch and Quasi-Newton) for every simulated path (Figure 4.9). Ideally, the best neural controller performance (in terms of having a smaller y-path error as compared to the optimal controller) would have the highest percentage of weight change. However, for lane change and sudden change of direction, the maximum path errors conflict with the obtained average weight change percentage. The neural controllers for both batch-training methods have better performances than the gradient-online method. The conflict is due to the fact that there were more epochs simulated for the online training method as compared to the batch training methods.

The remark on the average weight change percentage is also inapplicable for the sinus path. The neural controller with online training method has better performance than the batch training methods, although the percentage of average weight change of the former is smaller than the latter. The reason for this behaviour is that in some parameter space, the accumulated weight changes for batch training become large. As written in [8], this leads batch training to use unreasonably large steps, which subsequently results to unstable learning and to the overshooting of curves and local minima in the error landscape.

The use of batch training, for some paths, can improve the accuracy of the controller. Apart from that, most of the time, batch training involves less training time than the online training. These results are achievable as the network for the controller is small scaled (judged by its number of weights).

4.5 CONCLUSIONS

In this chapter, a neural network controller has been implemented and trained in three different conditions. Each of the three conditions has its own limitations and capabilities. While it might take longer training time with online training, the algorithm is able to find a good set of weights and achieves a global minimum. On the other hand, even if the batch training is proven to be faster and more accurate, it may not perform very well if the controller network is upgraded to a larger scale.

So far, the car model has been trained with a low speed of 20 m/s, with not so much effect on the lateral or yaw acceleration. In the next chapter, a new car model is introduced and it will be trained with a higher speed to yield a non-linear behaviour of the car.

CHAPTER 5

NEURAL NETWORK STEERING CONTROL OF A NONLINEAR CAR MODEL

This chapter is another advancement of works done by [2]. Previously a nonlinear system was controlled by a non-linear network trained using the online gradient method. Similar to the linear system discussed in the previous chapter, batch training (Gradient and Quasi-Newton) is introduced for comparison. Updated weights, learning rates and time taken after final epochs are considered in the study.

The first part of the chapter examines the learning processes using the gradient method (online and batch training modes). The second part of the chapter involves learning processes using the Quasi-Newton method. A comparison of average weight change percentage for the three types of training is highlighted in the third section. The next and final parts of this chapter discuss and conclude the observation for controlling a nonlinear system by nonlinear neural networks.

5.1 IMPLEMENTATION USING GRADIENT METHOD

5.1.1 Neural Network Controller Implementation

The controller is set to be a single processing neuron. As the system to be controlled is nonlinear, the activation function of the neural network is replaced from the MATLAB command 'purelin' (previously for linear system) to a tan-sigmoid function. A tan-sigmoid function will result in output value to fall within interval [-1,1]. This function originates from hyperbolic tangent function, which has the same shape. As speed is important in training neural networks, this function is a good trade off.

As in the linear network, the input to the controller is the augmented state z = [x $\quad$ y_r]^T. The output of the neuron is the steering wheel angle, δ_{sw} which is represented by:

$$\delta_{sw} = w_1(k).z_1(k)+w_2(k).z_2(k)+....w_{n+5}(k).z_{n+5}(k) \qquad (5.1)$$

The vehicle performance is evaluated using the linear quadratic cost function as in Chapters 3 and 4:

$$J(z(k), \delta_{sw}(k)) = z^T(k).R_1.z(k)+\delta_{sw}(k).R_2.\delta_{sw}(k) \qquad (5.2)$$

From the equation, the partial derivatives of the cost with respect to the augmented state ($\partial J(z(k),\delta_{sw}(k)/\partial z(k)$) and the partial derivatives of the cost with respect to control variable ($\partial J(z(k),\delta_{sw}(k))/\delta_{sw}(k)$) can be obtained. By knowing the previous derivatives at time kT, the derivatives of the augmented state and the control variable with respect to the weighting vector w at time (k+1)T can be determined.

The sensitivity matrices of Φ with respect to the state vector elements, (dΦ/dz(k)) and control variable (dΦ/dδ_{sw} (k)) have been included in [2], and will not be repeated in the report. The derivatives of the side forces with respect to the state vector elements and control variable can be obtained using the MATLAB function 'diff'.

As was done for the linear system, the initial weighting parameters W_0 for the neural controllers were taken from coefficients obtained from the optimal control theory.

5.1.2 Simulation by Online Training

The network is trained for different numbers of epochs depending on types of path. For some paths, small number of epochs for network training would be sufficient, but in some cases, it takes more epochs to improve the network performance. For all cases, the speed of the vehicle is set to 40 m/s.

A. Sinus Path (40 preview points)

Initially, after the first epoch, the maximum steady-state path error is 1.5×10^{-2} m (Figure 5.1(i)). By training the network up to three epochs, the maximum steady-state path error is reduced to 1.2×10^{-2} m (Figures 5.1 (ii)). The path errors during the first and final epochs are significantly less than the errors generated by the optimal controller. The learning rates become very small while the updated weights settle to some constant values after certain epochs (Figures 5.1(iii, iv)). The maximum steering wheel angle is 0.2 radians (Figure 5.1 (v). The observations for the sinus path are summarized in the following Table 5.1:

1.Sinus Path	
No of Epochs	3
Initial Learning Rate	0.008
Path Distance	900m
Maximum y-path Error (first epoch)	1.5×10^{-2} m
Maximum y-path Error (final epoch)	1.2×10^{-2} m
Final Learning Rate	2.869×10^{-78}
Learning Time (s)	300.231
Final Weight (at 10)	-0.8712

Table 5.1: *Summarized Observations for Sinus Path Following*

B. Lane Change (40 preview points)

Maximum y-path error decreases from $7x10^{-2}$m in the first epoch to $6x10^{-2}$m in the final epoch) using the neural controller. However, the maximum y-path error for the optimal controller is lower than the neural controller even after the final epoch (Figures 5.2(i, ii).

In terms of yaw attitude angle, the neural controller has better performance than the optimal controller (Figure 5.2 (iii)). Maximum steering wheel angle is 0.22 radians with neural controller as compared to 0.3 radians with optimal controller. The attitude angle following also has a better performance with the neural controller as compared to the optimal controller (Figure 5.2 (iv)). The learning rates become very small while the updated weights settle to some constant values after certain epochs (Figures 5.2(v, vi)). Table 5.2 below summarizes the whole observations:

2. Lane Change	
No of Epochs	15
Initial Learning Rate	0.05
Path Distance	300m
Maximum y-path Error (first epoch)	$7x10^{-2}$ m
Maximum y-path Error (final epoch)	$6x10^{-2}$ m
Final Learning Rate	1.472e-075
Final Weight (at 10)	-0.8716

Table 5.2: *Summarized Observations for Lane Change Path Following*

C. Sudden Change of Direction (40 preview points)

Judging from the maximum y-path error, the neural controller has better performance as compared to the optimal controller. The error reduces from $2.5x10^{-2}$ m at the first epoch to $2x10^{-2}$ m at the last epoch (Figures 5.3(i,ii)). The steering wheel angle is lower with the neural controller than with the optimal controller (Figure 5.3(iii)). Towards the final epoch, the learning rate decreases to some small values resulting in insignificant changes to the updated weights (Figure 5.3(iv, v)). The summary of the observation is as shown in Table 5.3:

3. Sudden Change of Direction	
No of Epochs	5
Initial Learning Rate	0.1
Path Distance	300m
Maximum y-path Error (first epoch)	$2.5\text{x}10^{-2}$ m
Maximum y-path Error (final epoch)	$2\text{x}10^{-2}$ m
Final Learning Rate	$6.5631\text{x}10^{-18}$
Final Weight (at 10)	-0.8713

Table 5.3: *Summarized Observations for Sudden Change of Direction*

D. Random Path (40 Preview Points)

The y-path error decreases from $5\text{x}10^{-3}$ m at the first epoch to $2\text{x}10^{-3}$ m at the final epoch using the neural controller. However, the errors are similar to the ones obtained using the optimal controller (Figures 5.4 (i, ii)). The learning rates and the updated weights posses similar behaviour as in previous cases (Figures 5.4 (iii, iv)). The summary of the observation is as shown in Table 5.4:

4. Random Path	
No of Epochs	3
Initial Learning Rate	0.1
Path Distance	900m
Maximum y-path Error (first epoch)	$5\text{x}10^{-3}$ m
Maximum y-path Error (final epoch)	$2\text{x}10^{-2}$ m
Final Learning Rate	$3.9622\text{x}10^{-85}$
Final Weight (at 10)	-0.8712

Table 5.4: *Summarized Observations for Random Path*

Figures 5.1: Sinus Path at 40m/s, 40 preview points

(Dashed: Optimal Controller, Solid: Neural Controller)

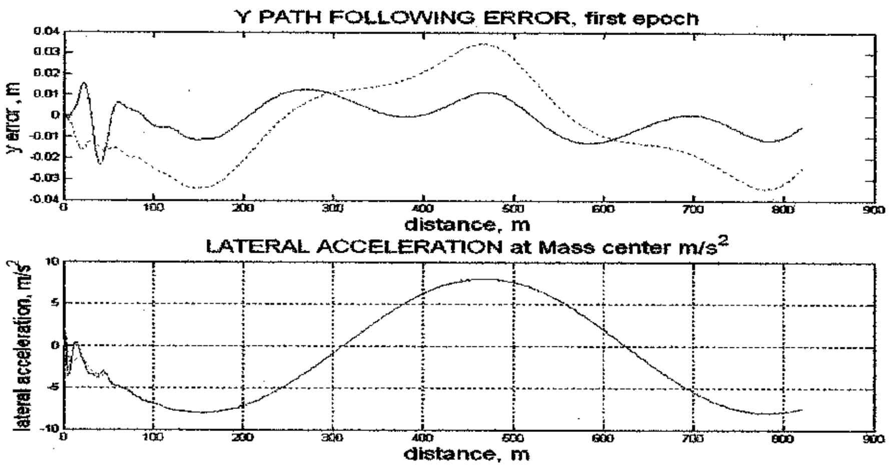


Figure 5.1(i) -top: *Plot of y-path following error at first epoch*
Figure 5.1(i) -bottom: *Plot of Lateral Acceleration at first epoch*

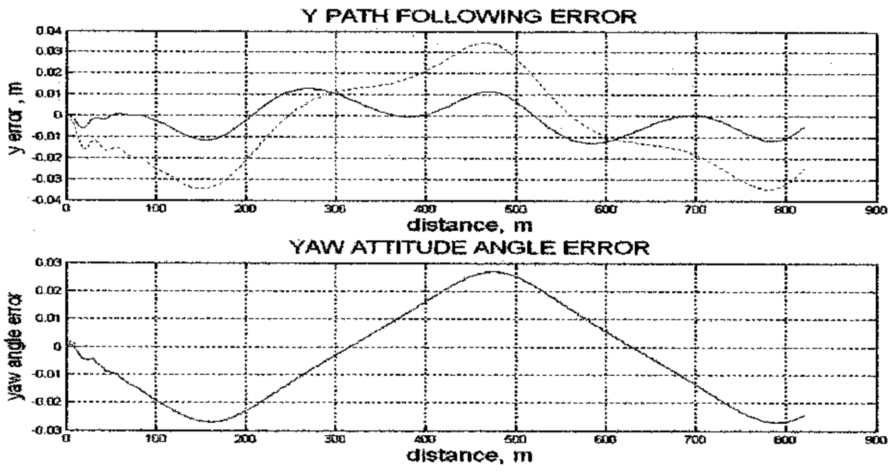


Figure 5.1(ii) -top: *y-path error at third epoch*
Figure 5.1(ii) - bottom: *Yaw Attitude Angle Error*

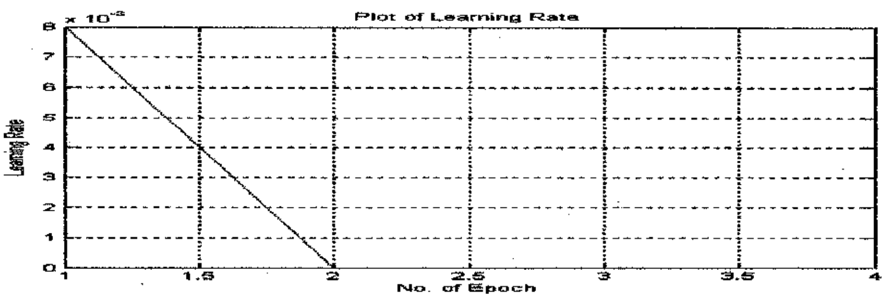


Figure 5.1(iii): *Plot of Learning Rate vs. No. Of Epoch*

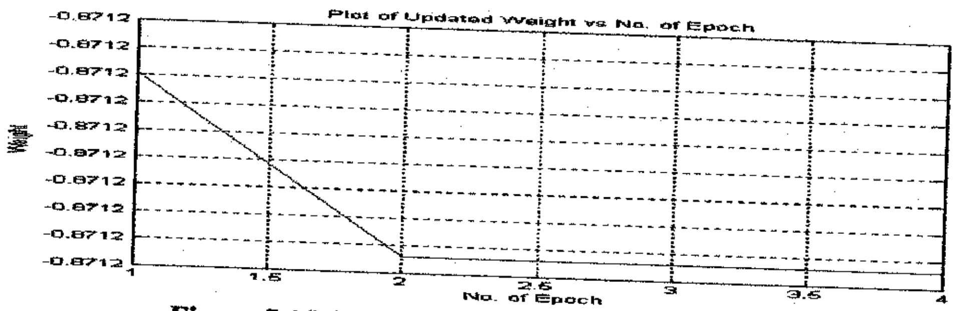
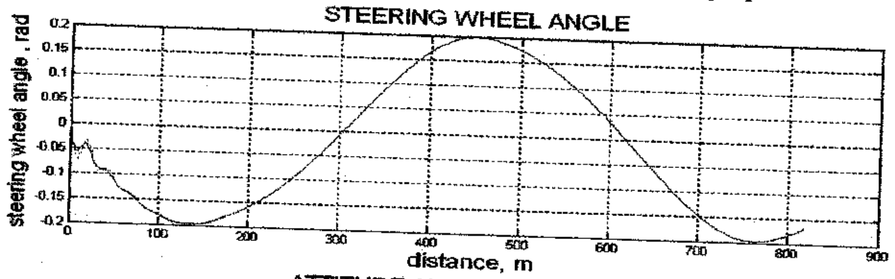


Figure 5.1(iv): *Plot of Updated Weight vs. No. Of Epoch*



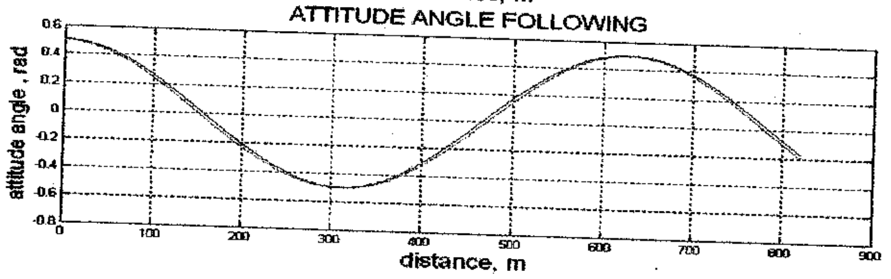


Figure 5.1(v) -top: *Steering Wheel Angle (Network Output)*
Figure 5.1(v) - bottom: *Attitude Angle Following*

Figures 5.2: Lane Change at 40m/s, 40 preview points
(Dashed: Optimal Controller, Solid: Neural Controller)

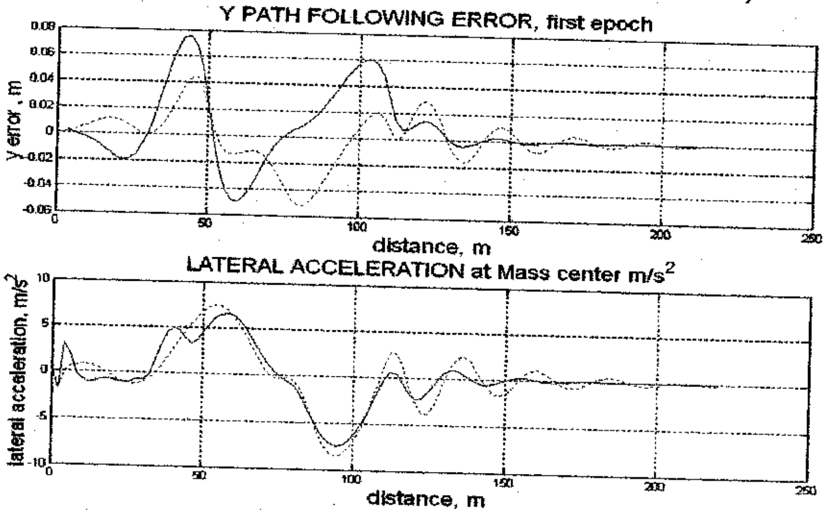


Figure 5.2(i) -top: *Plot of y-path following error at first epoch*
Figure 5.2(i) - bottom: *Plot of Lateral Acceleration at first epoch*

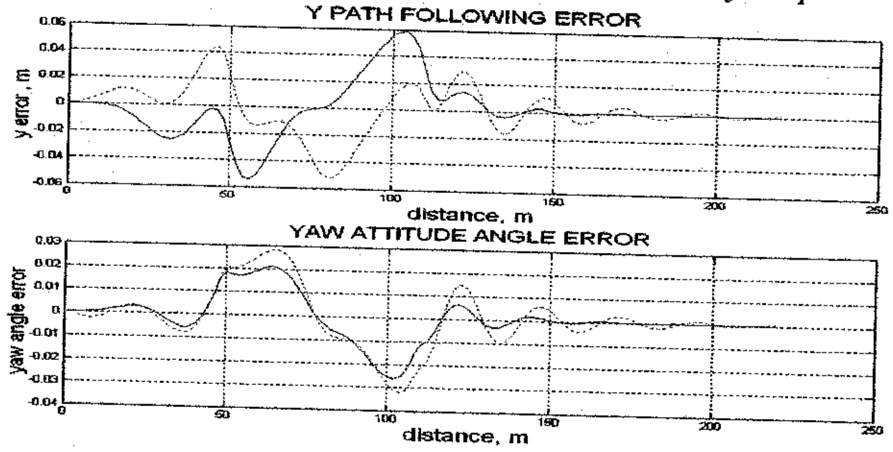


Figure 5.2(ii) - top: *y-path error at final epoch*
Figure 5.2(iii) -bottom: *Yaw Attitude Angle Error*

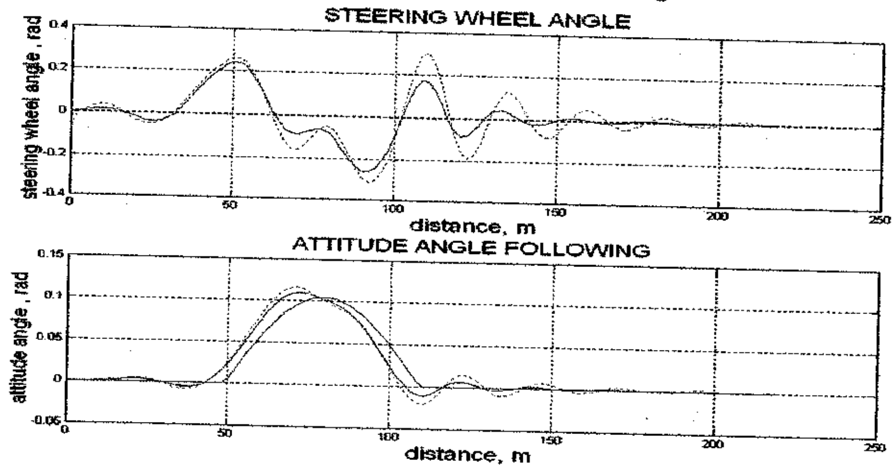


Figure 5.2(iv) -top: *Steering Wheel Angle (Network Output)*

Figure 5.2(iv) -bottom: *Attitude Angle Following*

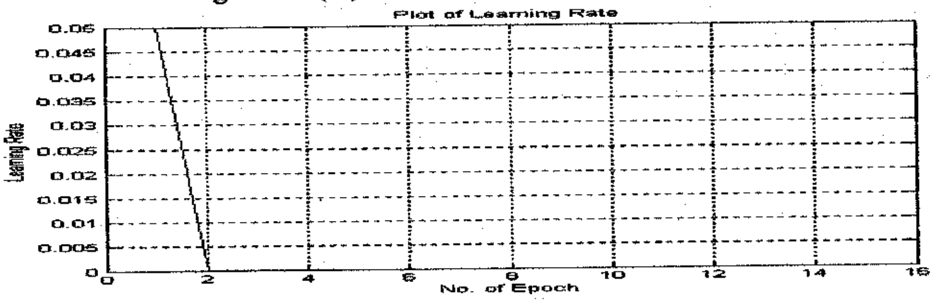


Figure 5.2(v): *Plot of Learning Rate vs. No. Of Epoch*

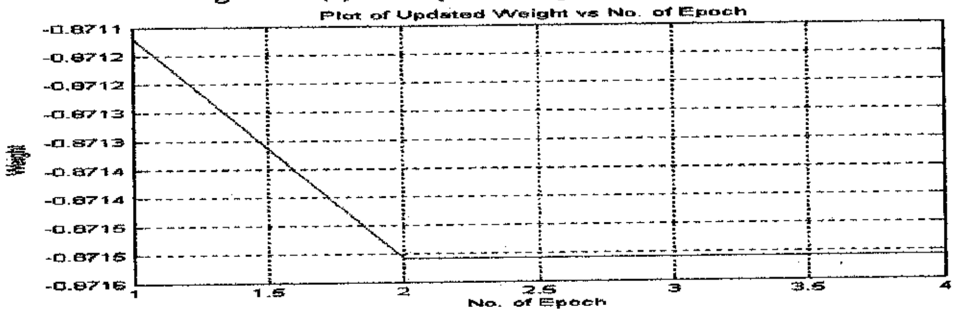


Figure 5.2(vi): *Plot of Updated Weight vs. No. Of Epoch*

Figures 5.3: Sudden Change of Direction at 20m/s, 40 preview points

(Solid: Neural Network, Dashed: Optimal Controller)

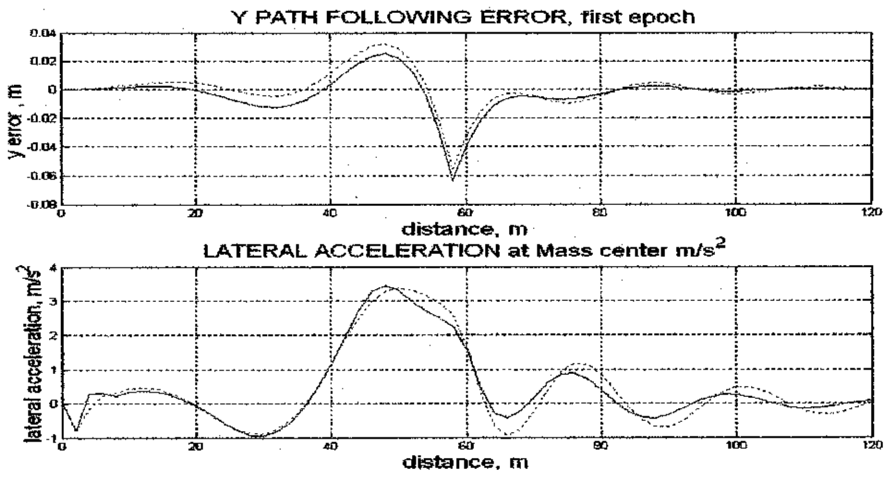


Figure 5.3(i) -top: *Plot of y-path following error at first epoch*
Figure 5.3(i) - bottom: *Plot of Lateral Acceleration at first epoch*

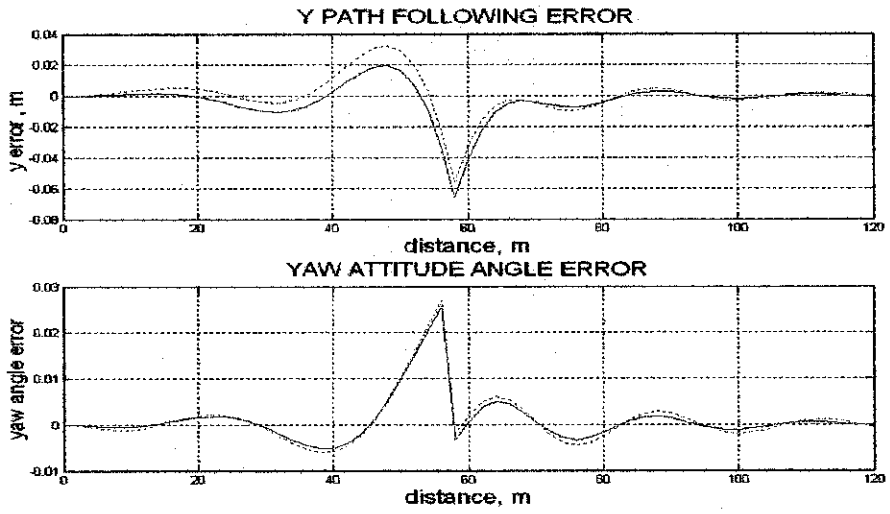


Figure 5.3(ii) -top: *y-path error at final epoch*
Figure 5.3(ii) -bottom: *Yaw Attitude Angle Error*

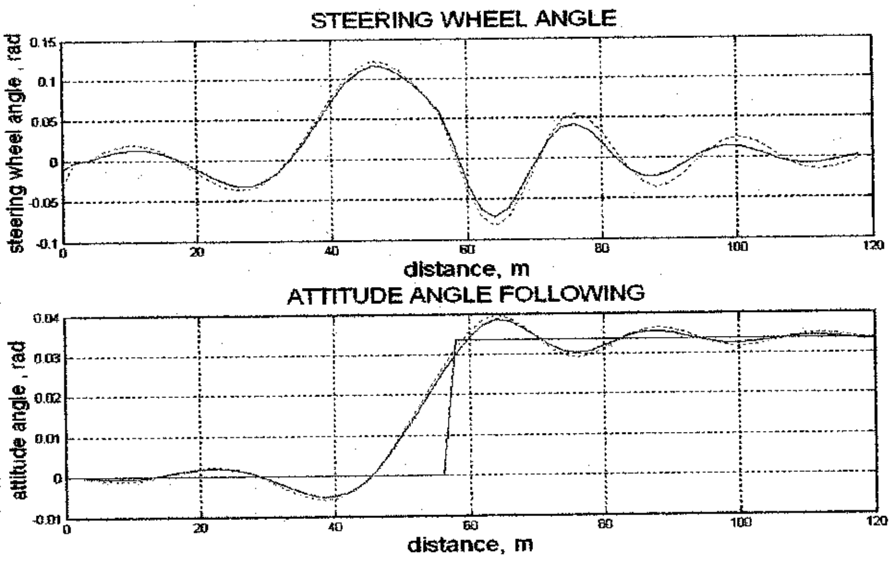


Figure 5.3(iii) -top: *Steering Wheel Angle (Network Output)*
Figure 5.3(iii) - bottom: *Attitude Angle Following*

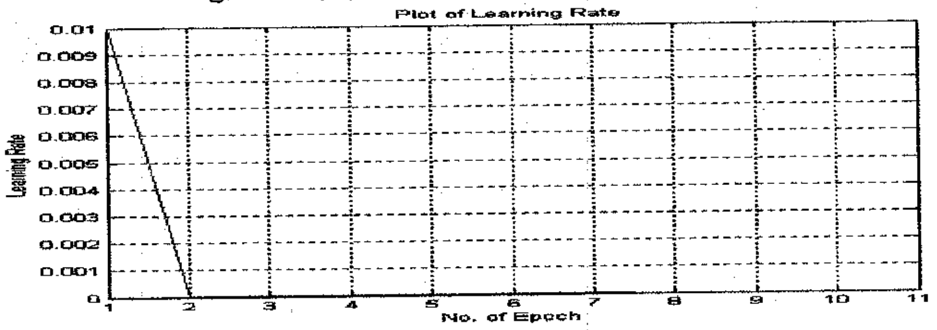


Figure 5.3(iv): *Plot of Learning Rate vs. No. Of Epoch*

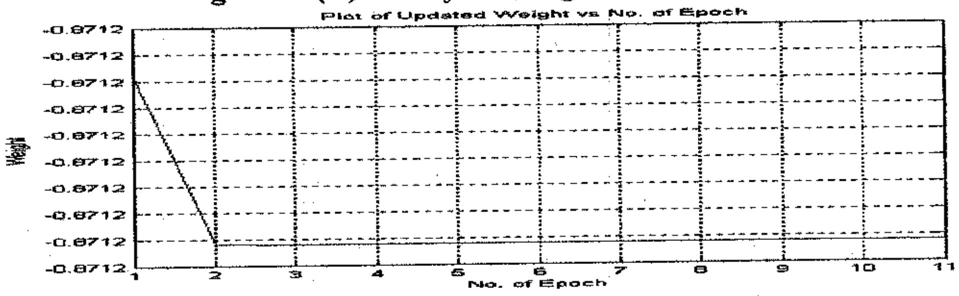


Figure 5.3(v): *Plot of Updated Weight vs. No. Of Epoch*

Figures 5.4: Random Path at 40m/s, 40 preview points

(Solid: Neural Network, Dashed: Optimal Controller)

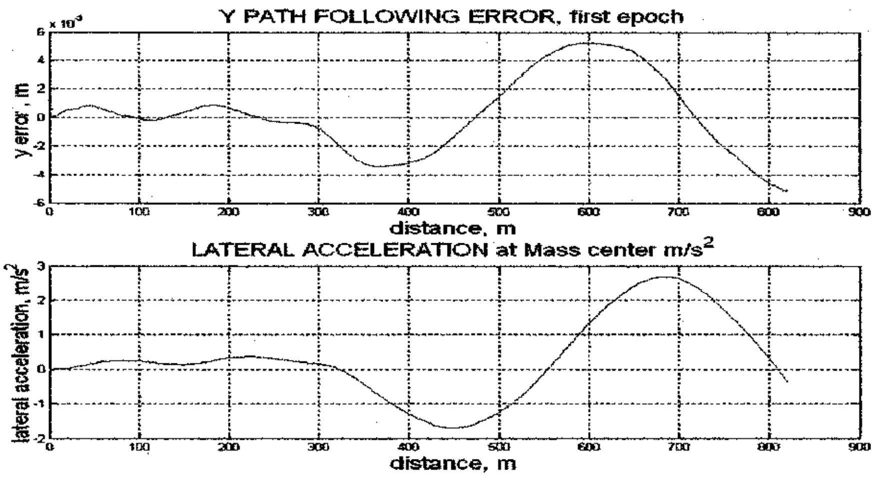


Figure 5.4(i) -top: *Plot of y-path following error at first epoch*
Figure 5.4(i) - bottom: *Plot of Lateral Acceleration at first epoch*

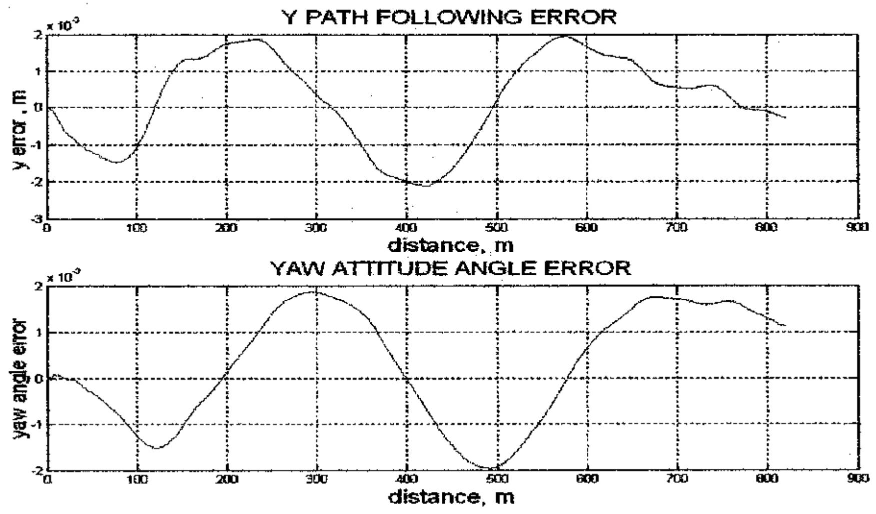


Figure 5.4(ii) - top: *y-path error at third epoch*
Figure 5.4(ii) - bottom: *Yaw Attitude Angle Error*

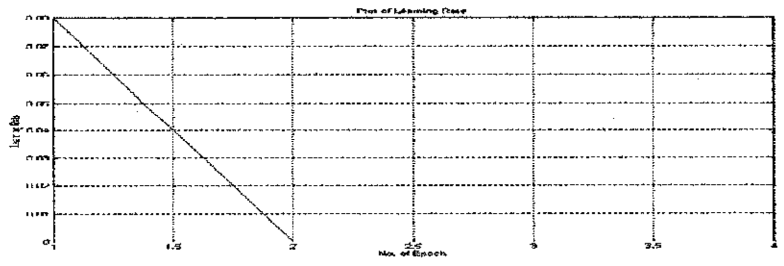


Figure 5.4(iii): *Plot of Learning Rate vs. No. Of Epoch*

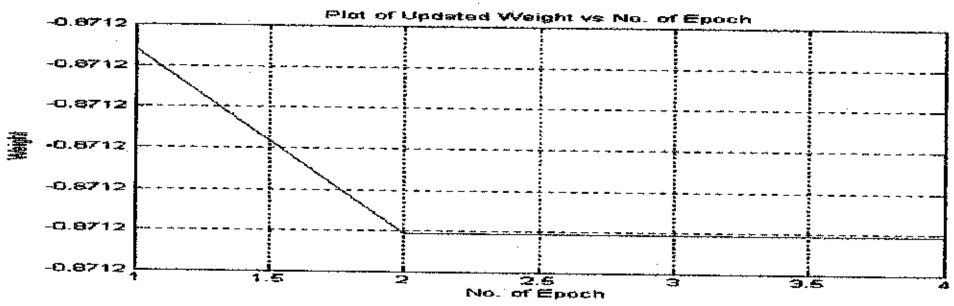


Figure 5.4(iv): *Plot of Updated Weight vs. No. Of Epoch*

4.1.3: Simulation by Batch Training

The simulation is set in the similar manner as in the linear system. The batch size is the total distance of path minus the number of preview points. The maximum number of epochs is 10, and the performance goal is $1\text{x}10^{-10}$. The preview points are arbitrarily set to 40 for all cases. The training will stop when the maximum number of epochs is reached or the performance goal is achieved.

For all four types of paths, the observations are similar. Although the obtained maximum y-path errors are small, the maximum errors are the same between the neural controller and the optimal controller. Apart from that, the performance goals are unachievable even if the maximum number of epochs is increased to 20. Tables and Figures 5.5-5.8 summarize and illustrate the observations obtained from the training.

1.Sinus Path	
Initial Learning Rate	0.008
Path Distance	900m
Maximum y-path Error	0.035 m
Performance	0.00108494
Epoch to reach target	Performance goal is not achieved
Learning Time (s)	41.029
Final Weight (at 10th Point)	-0.8712

Table 5.5: *Summarized Observations for Sinus Path Following*

2. Lane Change	
Initial Learning Rate	0.05
Path Distance	300m
Maximum y-path Error	$7\text{x}10^{-2}$ m
Performance	1.25186e-005
Epoch to reach target	Performance goal is not achieved
Learning Time (s)	11.587
Final Weight (at 10th Point)	-0.8711

Table 5.6: *Summarized Observations for Lane Change Path Following*

3. Sudden Change of Direction	
Initial Learning Rate	0.01
Path Distance	200m
Maximum y-path Error	0.055 m
Performance	1.30427e-006
Epoch to reach target	Performance goal is not achieved
Learning Time (s)	6.849
Final Weight (at 10th Point)	-0.87112

Table 5.7: *Summarized Observations for Sudden Change of Direction*

4. Random Path	
Initial Learning Rate	0.1
Path Distance	900m
Maximum y-path Error	$5\text{x}10^{-3}$ m
Performance	1.09739e-009
Epoch to reach target	Performance goal is not achieved
Learning Time (s)	47.138
Final Weight (at 10th Point)	-0.8712

Table 5.8: *Summarized Observations for Random Path*

Figures 5.5: Sinus Path at 40m/s, 40 preview points

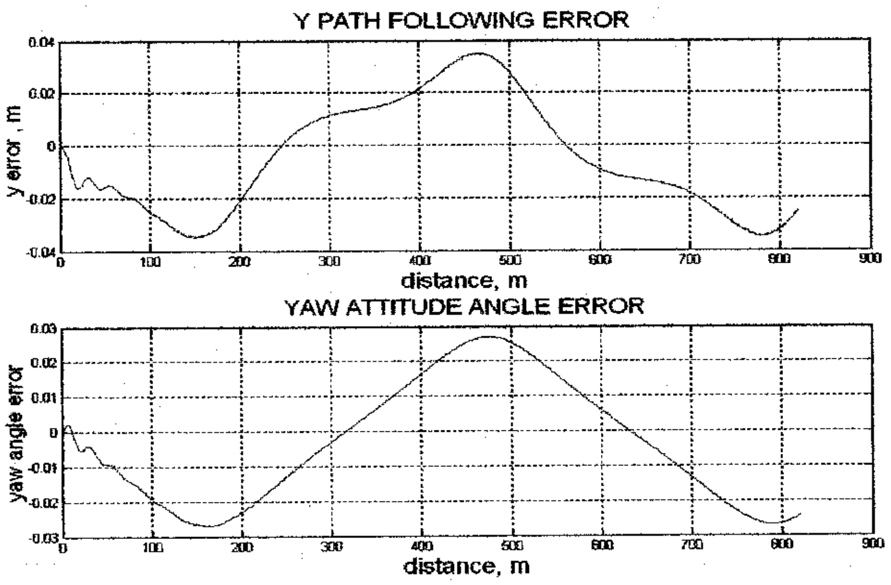


Figure 5.5(i) -top: *y-path error at final epoch*
Figure 5.5(i) - bottom: *Yaw Attitude Angle Error*

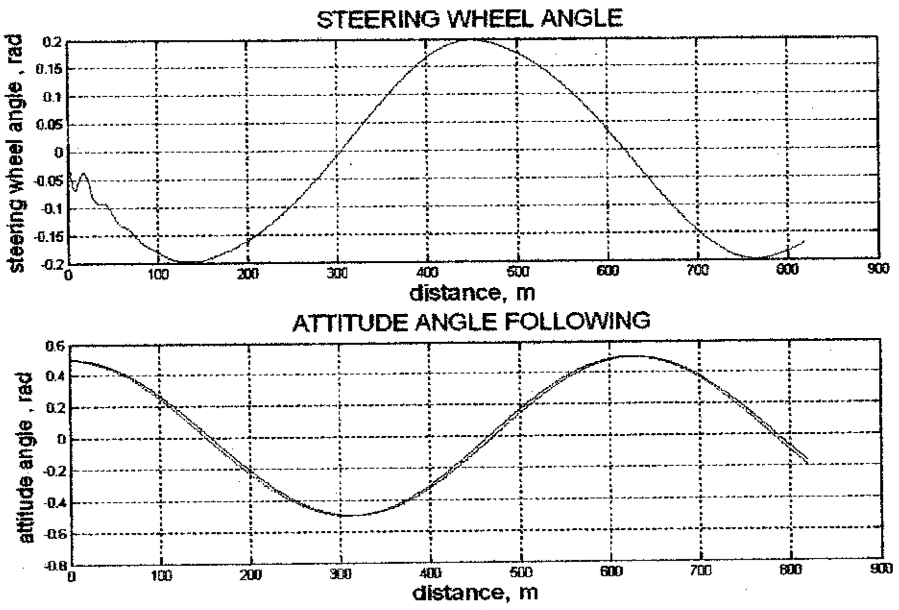


Figure 5.5(ii) - top: *Steering Wheel Angle (Network Output)*
Figure 5.5(ii) - bottom: *Attitude Angle Following*

Figures 5.6: Lane Change at 40m/s, 40 preview points

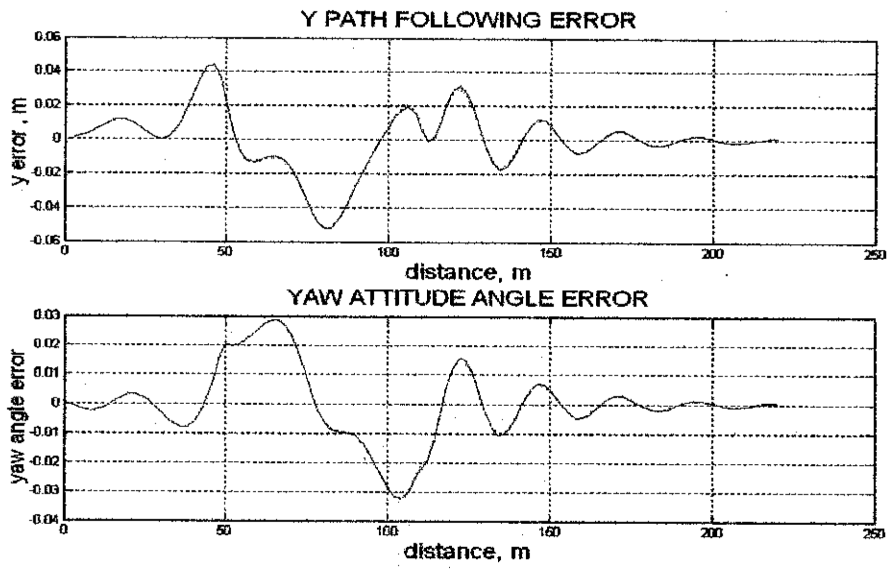


Figure 5.6(i) -top: *y-path error at final epoch*
Figure 5.6(i) -bottom: *Yaw Attitude Angle Error*

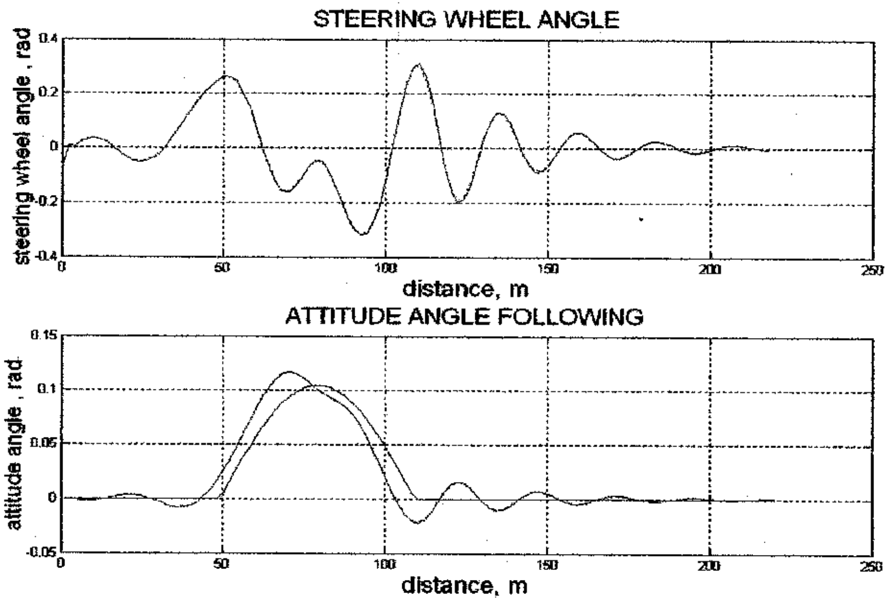


Figure 5.6(ii) - top: *Steering Wheel Angle (Network Output)*
Figure 5.6(ii) - bottom: *Attitude Angle Following*

Figures 5.7: Sudden Change of Direction at 40m/s, 40 preview points

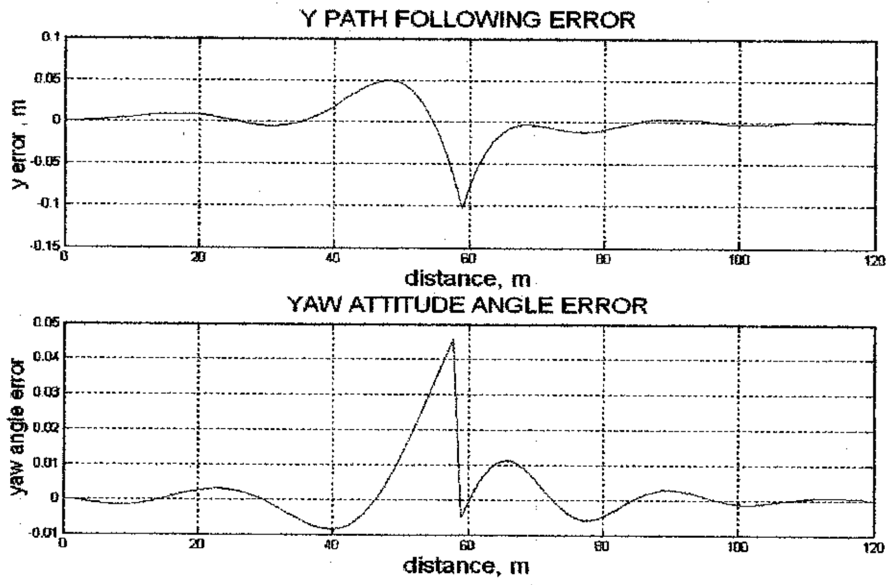


Figure 5.7(i) - top: *y-path error at final epoch*
Figure 5.7(i) -bottom: *Yaw Attitude Angle Error*

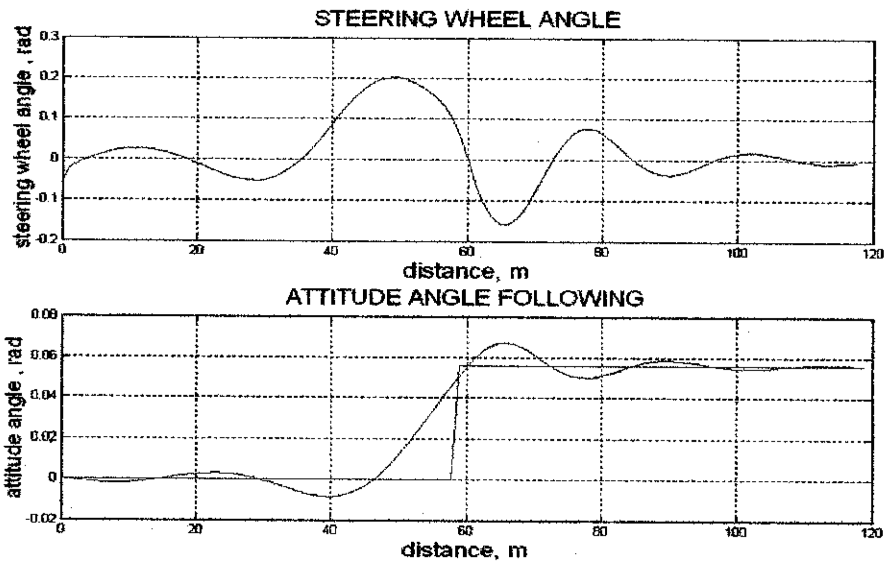


Figure 5.7(ii) - top: *Steering Wheel Angle (Network Output)*
Figure 5.7(ii) - bottom: *Attitude Angle Following*

Figures 5.8: Random Path at 40m/s, 40 preview points

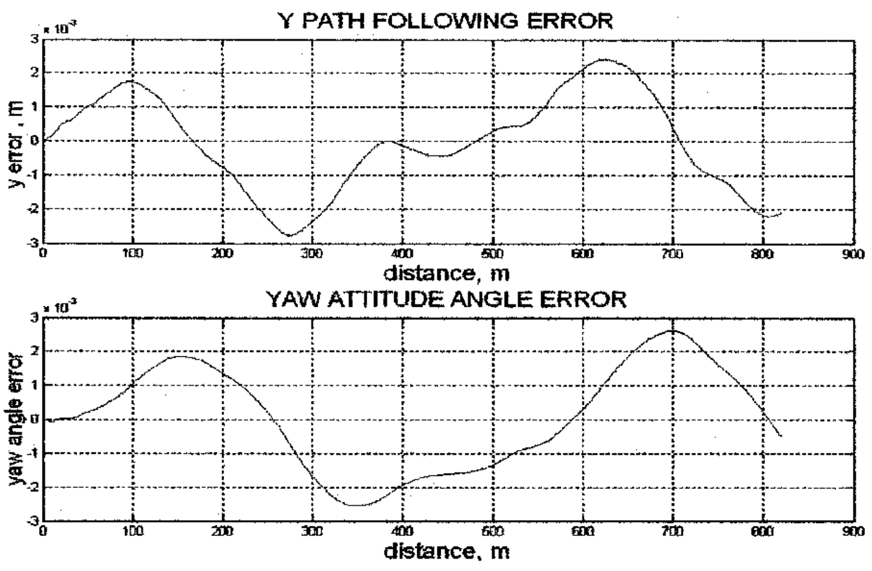


Figure 5.8(i) - top: *y-path error at final epoch*
Figure 5.8(i) - bottom: *Yaw Attitude Angle Error*

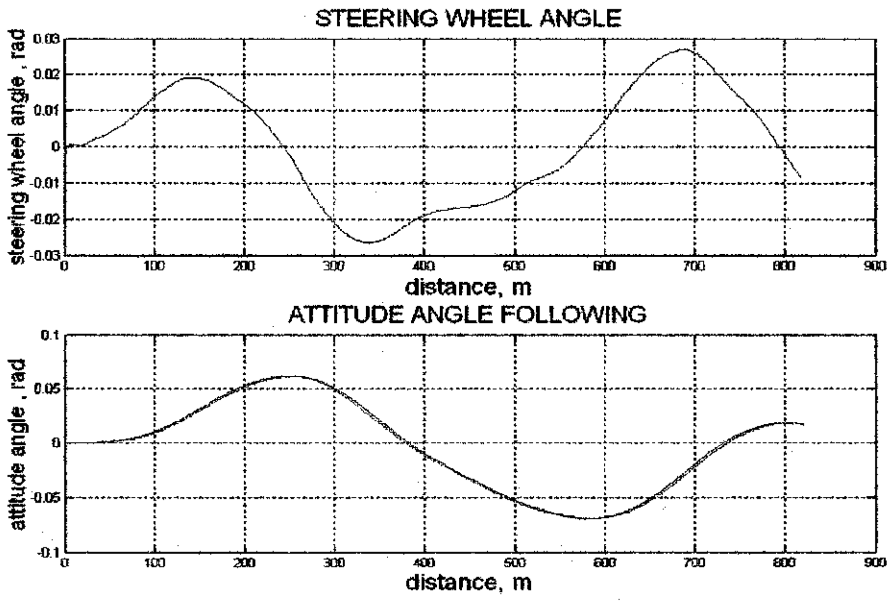


Figure 5.8(ii) - top: *Steering Wheel Angle (Network Output)*
Figure 5.8(ii) -bottom: Attitude Angle Following

5.2 IMPLEMENTATION USING QUASI-NEWTON METHOD

5.2.1 Neural Network Controller Implementation

To implement the controller, all details described in the previous section are adopted. The MATLAB command 'traingda' for the gradient-batch training is switched to 'trainbfg', which is the command for BFGS Quasi-Newton back propagation. According to the MATLAB toolbox [6], this command can train any network provided that its weights, net inputs and transfer functions have derivative function. Similar for the linear system in the previous chapter, the line search algorithm to locate the minimum point is the one-dimensional minimization using Backtracking method.

The maximum number of epochs is set to 10. One epoch is exactly one batch, which is equivalent to the path distance minus the preview points. For all four paths, the initial learning rates, initial weights, number of preview points, speed and path distances are similar to the previous cases. Similar to the previous batch training, the transfer function used for the network is tan-sigmoid.

5.2.2 Simulation Results

For all four types of paths, the observations are similar to the previous gradient-batch method. Although the obtained maximum y-path errors are small, the maximum errors are the same between the neural controller and the optimal controller. However by training the network using the Quasi-Newton method, the performance targets are achievable. Tables and Figures 5.9-5.12 summarize and illustrate the observations obtained from the training.

1.Sinus Path	
Initial Learning Rate	0.008
Path Distance	900m
Maximum y-path Error	0.035 m
Performance	2.5311e-015
Epoch to reach target	3
Learning Time (s)	47.178
Final Weight (at 10th Point)	-0.8712

Table 5.9: *Summarized Observations for Sinus Path Following*

2. Lane Change	
Initial Learning Rate	0.05
Path Distance	300m
Maximum y-path Error	$7x10^{-2}$ m
Performance	4.77988e-019
Epoch to reach target	1
Learning Time (s)	13.018
Final Weight (at 10th Point)	-0.8711

Table 5.10: *Summarized Observations for Lane Change Path Following*

3. Sudden Change of Direction	
Initial Learning Rate	0.01
Path Distance	200m
Maximum y-path Error	0.055 m
Performance	1.30427e-006
Epoch to reach target	1
Learning Time (s)	7.14
Final Weight (at 10th Point)	-0.87112

Table 5.11: *Summarized Observations for Sudden Change of Direction*

4. Random Path	
Initial Learning Rate	0.1
Path Distance	900m
Maximum y-path Error	$5x10^{-3}$ m
Performance	4.40456e-013
Epoch to reach target	2
Learning Time (s)	41.41
Final Weight (at 10th Point)	-0.8712

Table 5.12: *Summarized Observations for Random Path*

Figure 5.9: Sinus Path at 40m/s, 40 preview point

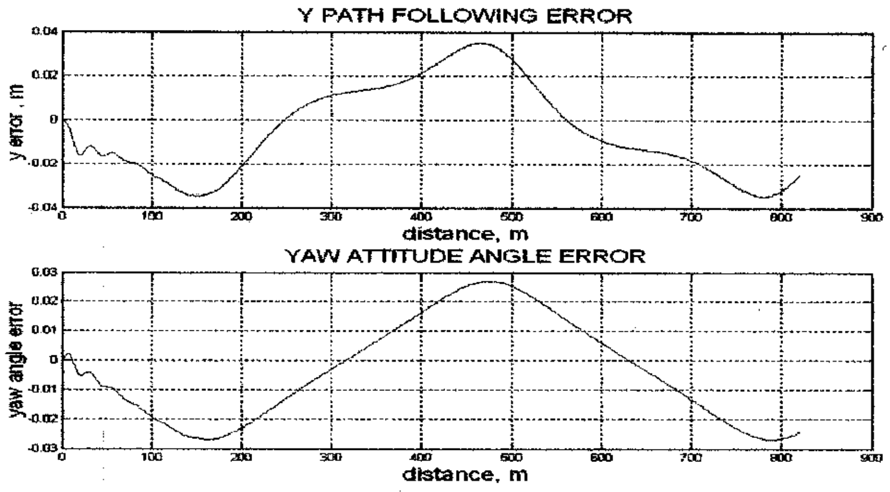


Figure 5.9(i) -top: *y-path error at final epoch*
Figure 5.9(i) - bottom: *Yaw Attitude Angle Error*

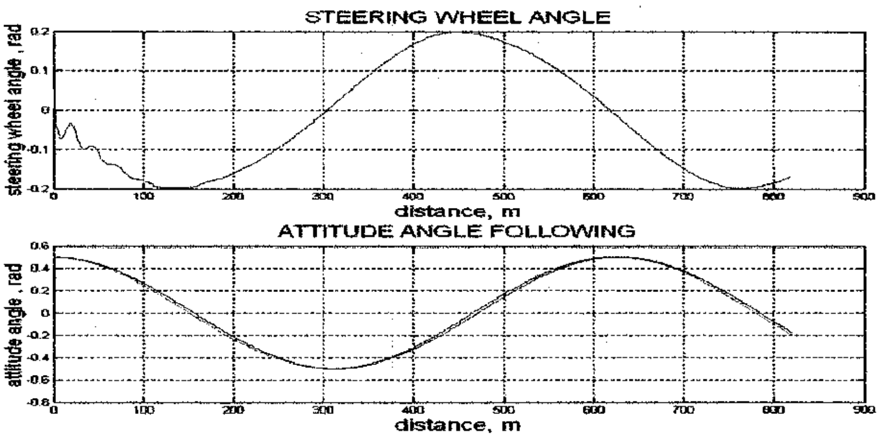


Figure 5.9(ii) -top: *Steering Wheel Angle (Network Output)*
Figure 5.9(i) - bottom: *Attitude Angle Following*

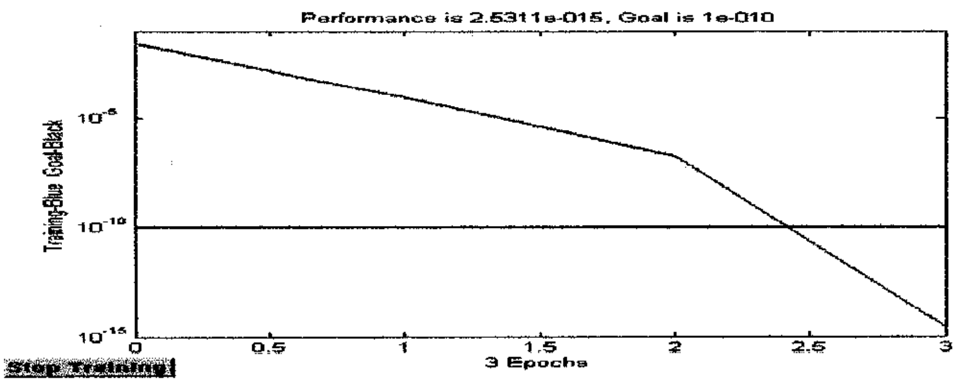


Figure 5.9(iii): *Training Result*

Figure 5.10: Lane Change at 40m/s, 40 preview point

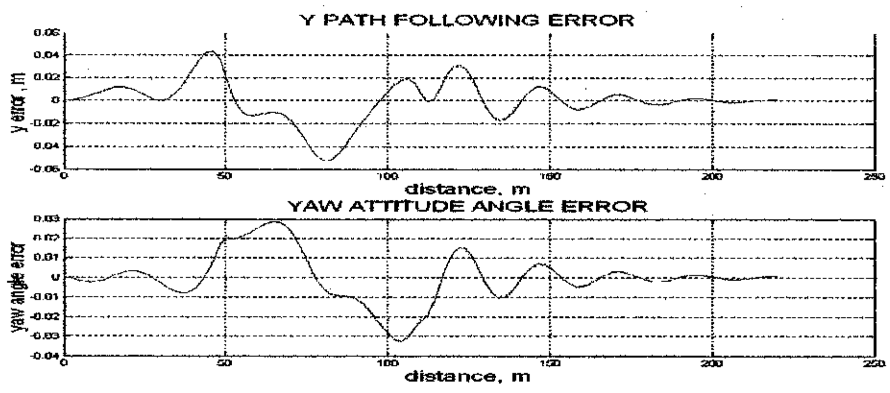


Figure 5.10(i) - top: *y-path error at final epoch*
Figure 5.10(i) - bottom: *Yaw Attitude Angle Error*

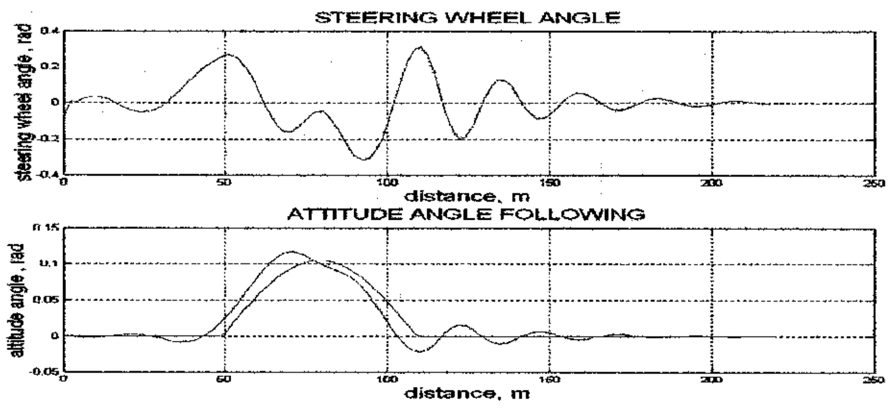


Figure 5.10(ii) - top: *Steering Wheel Angle (Network Output)*
Figure 5.10(ii) - bottom: Attitude Angle Following

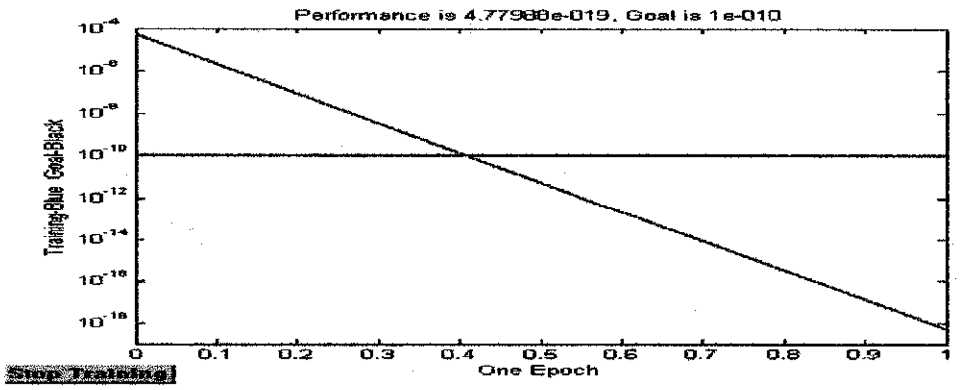


Figure 5.10(iii): Training Result

Figure 5.11: Lane Change at 40m/s, 40 preview point

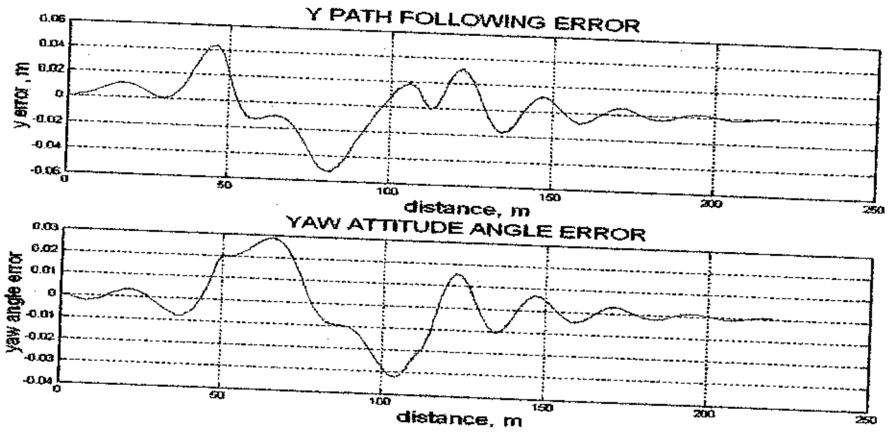


Figure 5.11(i) - top: *y-path error at final epoch*
Figure 5.11(i) - bottom: *Yaw Attitude Angle Error*

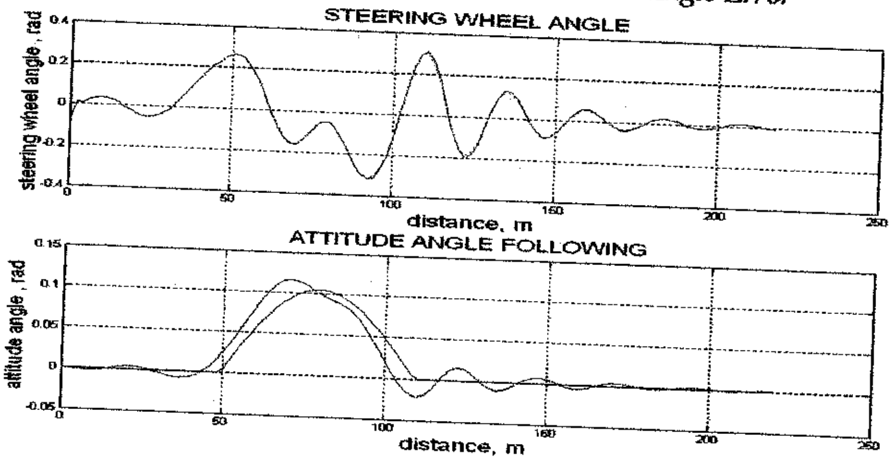


Figure 5.11(ii) - top: *Steering Wheel Angle (Network Output)*
Figure 5.11(ii) - bottom: *Attitude Angle Following*

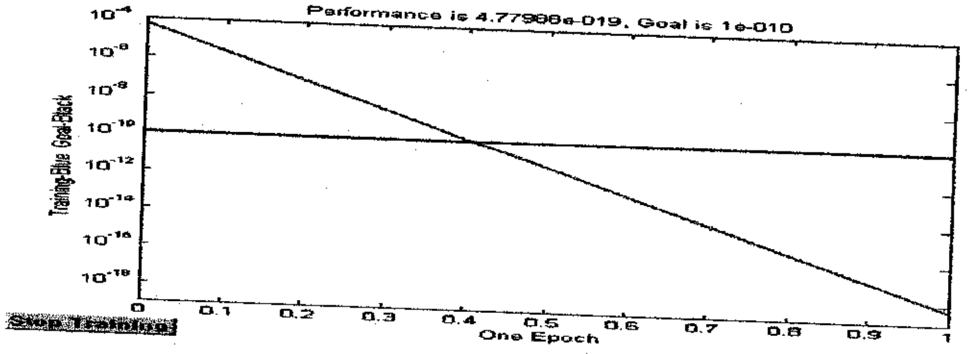


Figure 5.11(iii): *Training Result*

Figure 5.12: Lane Change at 40m/s, 40 preview point

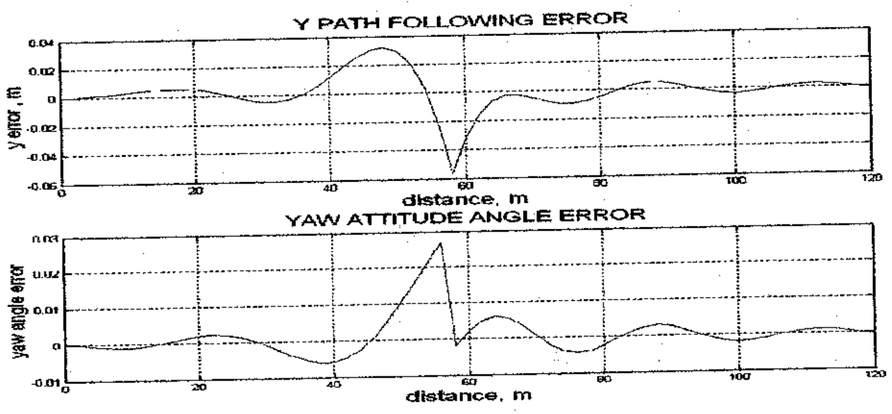


Figure 5.12(i) -top: *y-path error at final epoch*
Figure 5.12(i) - bottom: *Yaw Attitude Angle Error*

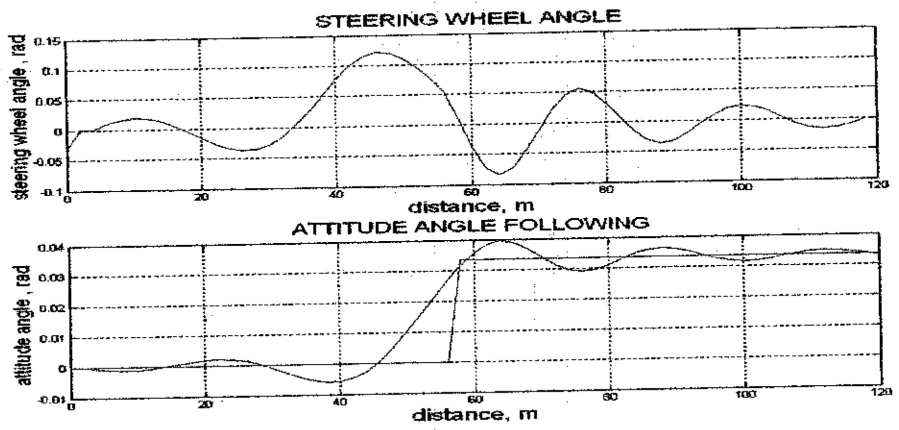


Figure 5.12(ii) - top: *Steering Wheel Angle (Network Output)*
Figure 5.12(ii) - bottom: *Attitude Angle Following*

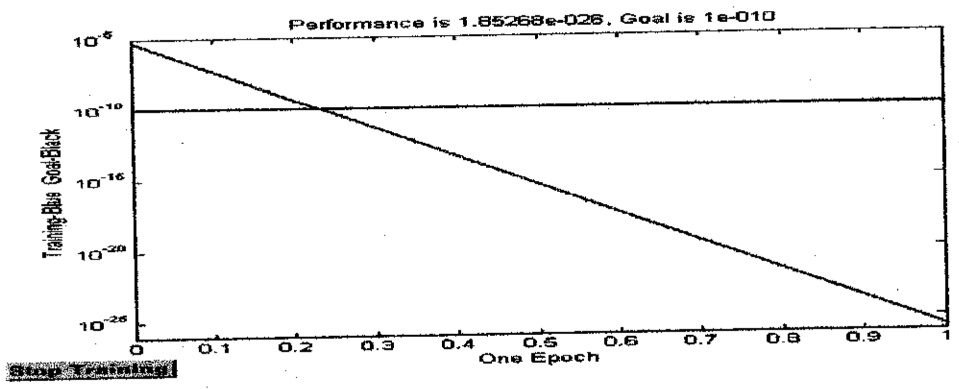


Figure 5.12(iii): Network Output

5.3 PERCENTAGE OF AVERAGE WEIGHT CHANGE

To determine weight changes between the original weights obtained through the linear optimal control theory and the weights at the final epochs, the percentages of average weight change were calculated. The obtained percentages are useful in determining whether the networks have experienced 'good' or 'bad' learning. Learning is considered 'good' if the obtained percentage is high and the neural controller has better performance than the optimal controller. On the other hand, learning is 'bad' when the percentage is high but the performance of the neural controller is similar to or worse than the optimal controller.

The calculation formula is similar as with the linear systems:

$$\text{Average Weight Change} = \frac{\sum_{i=1}^{n} \left| \left(\frac{W_{NN(i)} - W_{OC(i)}}{W_{OC(i)}} \right) \right| x100\%}{n} \qquad (5.3)$$

where W_{NN} is the updated weight by neural network controller

W_{OC} is the original weight of the optimal controller, and

n is total number of weights

The summarized calculation of the percentage is as shown in the following Table 5.13:

	Quasi-Newton (%)	Gradient (Batch) (%)	Gradient (Online) (%)
1) Sinus	10.4643	10.3690	10.5278
2) Lane Change	5.3685	4.5638	37.7701
3) Sudden Change	2.3992	2.2801	25.7712
4) Random Path	8.8116	12.5952	2.9375

Table 5.13: *Nonlinear Car Model: Percentage of Average Weight Change*

5.4 DISCUSSIONS

From the obtained plots and percentage calculations, it seems that network training by batch method is not really suitable for nonlinear systems. According to Barto [12], nonlinear models can generate error surfaces with many local minima. This makes it impossible for the network to achieve global convergence. Linear systems, on the other hand, do not face this complexity because no matter what fixed presentation is used, its mean square error is a quadratic function of the parameters with a unique minimum.

Another reason that may have contributed to inability of batch training to produce better results is due to the behaviour of the method of training itself. In batch training, large accumulated weights after one epoch can lead to unreasonably large steps. This in turn will result in unstable learning and to the overshooting of curves and local minima in the error landscape.

In contrast to the observation obtained in the previous chapter, the percentages of average weight change tally with the observed plots. Higher percentage indicates better performance of the neural controller. This means that the particular network has experience a 'good' training, as have been observed with the lane change and sudden change of direction.

However, for sinus path, although the percentages of average weight change vary a little between the three modes of training (gradient – online, gradient – batch and Quasi-Newton), the controller performances differ. While the maximum y-path error generated by the neural controller is less than the one generated by the optimal controller using the online training method, there seems to be no network learning with the batch mode. The maximum y-path errors between both controllers are the same. This is because for an oscillating type of path, network training in batches accumulates large weight, which results in inaccuracy to follow curving gradient throughout each epoch. This makes learning become inefficient. While it is safer to use a higher learning rate for online training, the controller performance generated by batch training will suffer even more.

Another observation that is worth mentioning is the inability of the vehicle to follow the path for speed greater than 40 m/s. Many trials on reduction of learning rate, reduction of sample time and increment of preview points have been done, but no improvement was achieved. A sensible solution for this matter is probably to introduce a multilayer network that incorporates tan-sigmoid transfer function in hidden layer and linear transfer function at the output layer. This will make the network become more capable with nonlinear system, and the network outputs can take on any value without limitation to any range.

As observed in the previous chapter, the behaviour of the updated weight is parallel with the behaviour of the learning rates. Depending on type of path, the learning rates will either jump up or oscillate, before decreasing rapidly in the first epoch. In the successive epochs, the rates decrease slowly, having little influence on the updated weights. This means that as training progresses, the weight changes will eventually settle to a constant value.

5.5 CONCLUSIONS

In this chapter, the implementation and the simulation results of a single processing neuron to control a nonlinear car model have been discussed thoroughly. Similar to the linear system, the controller has been trained in three different conditions and compared to the optimal controller.

The simulation by online training gives a better performance than the optimal controller in terms of maximum y-path errors, maximum steering wheel angle and yaw attitude angle error. On the other hand, although simulation by batch training produces acceptable results and shorter training time, there are no performance improvements when being compared with the optimal controller.

The maximum allowable speed to ensure the vehicle follows the paths in both modes of training is considerably low. It might be possible to implement a mechanism that allows the network to reduce speed when the vehicle could not follow curvatures and sharp turns, and return to the original speed when the path is smoother.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 CONCLUSIONS

This thesis emphasized comparisons of neural controllers trained in two different modes: online training (Gradient method) and batch training (Gradient and Quasi-Newton methods). The neural controllers were implemented to operate both linear and nonlinear cars, moving on simulated paths. The study also puts much exposure on behaviour of learning rates and updated weights.

The capabilities and limitations of the two modes of training depend on factors such as vehicle type (linear or nonlinear), type of path, size of learning rate as well as number of epochs. A controller that is trained in batch mode can perform really well in a linear system in such a way that it produces smaller maximum errors (as compared to the optimal controller) and shorter training time (as compared to online training).

On the other hand, in nonlinear system, the capability of online training surpasses the capability of batch training. The neural controller trained by online training has smaller maximum errors than the optimal controller. The batch training experienced 'bad learning' in nonlinear system because the performance of the neural controller remains similar as with the optimal controller, even though the network weights are updated and changed throughout the epochs. To date, the implemented neural controller is still unable to deal with the nonlinearity of the car regardless of different algorithms used (Gradient and Quasi-Newton methods).

For both linear and nonlinear systems, the controller performances depend heavily on suitable selection on learning rates, which enable the updated weights to converge to the best minimum.

6.2 PROPOSALS FOR FURTHER WORK

Several recommendations on future works for expansion and continuation of the project are as follows:

6.2.1 Additional neural control of the forward speed

This additional feature will enable the car to move in non-constant speed. This way, the network will reduce the velocity of the car when moving at sharp curves or turns and return to the original velocity when the path is smoother. Thus better path following will be achieved.

6.2.2 Improvement of neural network efficiency

The improvement could be achieved by adding extra layers to the network. Although this addition will increase the network's complexity, it will probably work very well, because, as written by Tsoukalas and Uhrig [13], the multi-layer networks have greater representational power than the single-layer network for nonlinear systems. Apart from that, different search routines for the Quasi-Newton method could be tried out to improve the efficiency of the network.

6.2.3 Implementation of different learning process

So far, two types of learning process have been tried out: Gradient descent and Quasi-Newton method. It is possible to use other different learning process to improve the performance of the controller such as conjugate gradient method or Newton's method.

6.2.4 Implementation of different types of path

It would be interesting to see the car models able to follow paths that have obstacles such as holes on the road, children crossing the road or heavy truck ahead of the car, to name a few.

Taken from [2], other opportunities for further research may include:

- Car model could be improved by decoupling right and left wheels on one axle as well as considering aerodynamic forces.
- Other parameters for performance index J could be introduced such as lateral or yaw acceleration.
- The sample time and the number of preview points can be decoupled to allow a suitable selection of preview points.

References

[1] R.S. Sharp, V. Valtetsiotis, *Optimal Preview Car Steering Control*, Supplement to Vehicle System Dynamics, 35 (P. Lugner and K Hendrick eds), May 2001, 101-117.

[2] Dandre, P., MSc. Thesis: *Learning Path Following Control of An Automobile*, Electrical Engineering Department, Imperial College London, 2003.

[3] Demuth, H.B., Beale, M., Neural Network Design, PWS Publishers, Boston, 1996, p 133.

[4] http://www.accpc.com/nnfaq/FAQ2.html

[5] Bertsekas; Tsikiklis, *Neuro-Dynamic Programming*, Ch. 3, Belmont, MA: Athena Scientific, 1996.

[6] Neural Network MATLAB Toolbox, Version 6.5

[7] Astolfi, A., Lecture Notes on *Optimization*, Electrical Engineering Department, Imperial College London, 2004.

[8] Wilson, R., Martinez, T.R., *The General Inefficiency of Batch Training for Gradient Descent Learning*.

[7] http://www.statsoft.com/textbook/steunet.html#gathering

[10] N. Louam, D Wilson and R.S Sharp, *Optimal Control of a Vehicle Suspension Incorporating the Time Delay Between Front and Rear Wheel Inputs*, Vehicle System Dynamics, 17(6), 1988, 317-336

[11] G. Prokop & R.S. Sharp, Performance Enhancement of Limited Bandwidth Active Automotive Suspensions By Road Preview. IEE Proceedings Control Theory and Applications 142(2), 1995, 140-148.

[12] Barto,A. G., *Connectionist Learning for Control,* Neural Network for Control, The MIT Press, 1992, 3rd Ed., pp 20.

[13] Tsoukalas, L.H, Uhrig R.E. 1997, *Fuzzy and Neural Approaches in Engineering,* New York, John Wiley & Sons

APPENDIX 1: MATLAB CODE

Neural Control of Linear Car Model (Online Training)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                LinearNN.m              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('  ----------------------------------------------------------------')
disp('   One Single Processing Element, Linear Car Model  ')
disp('  ----------------------------------------------------------------')
disp('')
clear all;close all;clc

% forward speed
   u=input('which speed ?  (using 20 par (default))');
   if isempty(u), u=20; disp('Using u=20m/s (default)'), end

% sampling period T
   T=0.05;

% number of preview points
   n=input('how many preview points (using 20 par (default))');
   if isempty(n), n=2/T; disp('Using a number corresponding to 1sec ahead (default)'), end

%car parameters definition
Cf=120000;
Cr=80000;
a=0.92;
b=1.38;
M=1200;
G=17;
Iz=1500;

%%%%%%%%%%%%%%%%%%%%%% Road Model Matrices%%%%%%%%%%%%%%%%%%%%

D=[zeros(n,1) eye(n); zeros(1,n+1)];
E=[zeros(n,1); 1];
%E=[0;0;0;0;0;0;0;0;1];

%%%%%%%%%%%%%%%%%%%%%%%Linear Car Model
%%%%%%%%%%%%%%%%%%%%%%%
Linear_car_model
disp('')

%%%%%%%%%%%%%%%%%%%%% Linear control gain calculaton
%%%%%%%%%%%%%%%%%%%
%cost prioritites (Priority is on PATH FOLLOWING)
Q=[100 0 ;
   0  1 ];
R2=1;
%compute the LQG gain Kt
LQRgain
```

```
%%%%%%%%%%%%%%%%%%%%%%%%Linear cost parameters
%%%%%%%%%%%%%%%%
%the cost to be minimised is the folowing one :
%J=Z(:,k)'*R1cost*Z(:,k)+delta(k)'*R2cost*delta(k)

R1cost=R1;
R2cost=R2;

tic      % Start a stopwatch timer
disp('         Loading path information .....')

%%%%%%%%%%%%%%%%%% %%%Path information
%%%%%%%%%%%%%%%%%%%%%%
for epoch = 1:5                % Setting iteration to 5 times
   if epoch == 1
   circuits_2
   else
   circuit_iterations
   end

[K,nb] = size(yref')   % Array size for yref'

%%%%%%%%%%%%%%  State definition & initialisation %%%%%%%%%%%%%%%%%%
% At each tiime step, a new global frame is defined
% The state is based on a frame comprising the local x and y-axes of the vehicle

%    Z=[  local lateral displacement v   ]
%      [  vdot                          ]
%      [  local angle phi                ]
%      [  phidot                        ]
%      [  local lateral preview errors   ]

% The notations A and B represents the optimal controller and
% the single processing element  respectively.

ZA = zeros(4+n+1,K-n-1);
ZA(1,1) = yref(1);
ZA(3,1) = (yref(2) - yref(1))/(u*T);
ZA(4+1:4+n+1,1) = yref(1:n+1)';

ZB = zeros(4+n+1,K-n-1);
ZB(1,1) = yref(1);
ZB(3,1) = (yref(2) - yref(1))/(u*T);
ZB(4+1:4+n+1,1) = yref(1:n+1)';

%augmented E matrx

Ebis=[zeros(4,1); E];

%%%%%%%%%%%%%%%%%% Paramaters Initialisation %%%%%%%%%%%%%%%%%%%
   %sensitivity functions initialized to 0
      dzdw = zeros(n+5,n+5);   %
      dudw = zeros(1,n+5);     %
```

```
    dJdw = zeros(1,n+5);        % to be multiplied with gama to obtain deltaw for gradient
mtd
    prevdJdw = zeros(1,n+5);
    deltaw = zeros(1,n+5);      % to be added to w to obtain w(k+1)
    prevdeltaw = zeros(1,n+5);

     %other parameters
    phiA(1)=(yref(2)-yref(1))/(u*T);
    phiB(1)=(yref(2)-yref(1))/(u*T);

    deltaA(1)=0;
    deltaB(1)=0;

    lateral_accelerationA(1)=0;
    lateral_accelerationB(1)=0;

    global_positionA(1)=ZA(1,1);
    global_positionB(1)=ZB(1,1);

    ZinitA = zeros(4+n+1,1);
    ZinitB = zeros(4+n+1,1);

    ZstepA = zeros(4+n+1, 1);
    ZstepB = zeros(4+n+1, 1);

 %%%%%%%%%%%%%%%% Neural network implementation%%%%%%%%%%%%%%%%
 disp('          neural network implementation.....')

 %choose an input layer with n+4 (number of states) neurons
 input=[-50*ones(n+5,1) 50*ones(n+5,1)];

 %net=newff(input,1,{'tansig'});
 net=newlin(input,1);

      %initialize the vector W(:) containing all weights and biases.
       if epoch ==1
          for jg=1:4+n+1
          W(jg)=Kt(jg);         % Weight based coeff obtained from optimal ctrl theory
          W_init=W;            % Storing the initial weight
          end
          %fixed learning rate
          gama=0.1;
          gama_init=gama;           % Storing the initial learning rate
          gama_next(1)=gama;
       else
          W = W_last;              % Last Updated Weight from Previous Epoch
          gama=gama_last;            % Last Updated Learning Rate from Previous Epoch
          gama_next(1) = gama;
       end
      %initialize neural network weightings
        net.IW{1,1}=W;
        net.b{1} =[0];
  toc   % reads the stopwatch timer
```

```
disp('            main loop.....')
tic    % starts another stopwatch timer

                    %%%%%%%%%%%%%%%%%%%%%%%%%%%% %
                    %                 MAIN LOOP                %
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k = 1:K-n-1

  % definition of a new global frame based on local x and y axes of the car

    % definition of the states of the car

      ZinitA = ZA(:,k);
      YdotA = ZA(2,k);

      ZinitB = ZB(:,k);
      YdotB = ZB(2,k);

        if k>1
              ZinitA(2) = ZinitA(2)-u*sin((phiA(k)-phiA(k-1)));  %the local y-axis
changed
          ZinitB(2) = ZinitB(2)-u*sin((phiB(k)-phiB(k-1)));
          %ZinitC(2) = ZinitC(2)-u*sin((phiC(k)-phiC(k-1)));
        else
          ZinitA(2)= 0;
          ZinitB(2)= 0;
          %ZinitC(2)= 0;
        end

    % due to the choice of the frame, absolute positions become zero
      ZinitA(1) = 0;
      ZinitA(3) = 0;
      ZinitB(1) = 0;
      ZinitB(3) = 0;

    % absolute to relative road data transformation

      local_yrefs = yref(k:k+n+1);

        for j = 1:(n+2),
            local_yrefsA(j) =  local_yrefs(j) - global_positionA(k)- ...
            (j-1)*phiA(k)*u*T;
            local_yrefsB(j) =  local_yrefs(j) - global_positionB(k)- ...
            (j-1)*phiB(k)*u*T;
          end

     % definition of the remaining states (preview path errors)
        ZinitA(4+1:4+n+1) = local_yrefsA(1:n+1);
        ZinitB(4+1:4+n+1) = local_yrefsB(1:n+1);


  % %%%%%%%%%%%%%%%% %%%%state error
  %%%%%%%%%%%%%%%%%%%%%
```

```
     epsA=ZinitA;
     epsB=ZinitB;
   %%%%%%%%%%%%%%%%%%%% steer angle
%%%%%%%%%%%%%%%%%%%%%%%
     deltaA(k) = -Kt*epsA;
     deltaB(k) = sim(net,-epsB);

   %state update

     ZstepA =  A *ZinitA+ B*deltaA(k) + Ebis*local_yrefsA(n+2);
     ZstepB =  A *ZinitB+ B*deltaB(k) + Ebis*local_yrefsB(n+2);

                         %%%%%%%%%%%%%%%%%%%%%%%%%%%
                         %            Weighting update            %
                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %dudw(k) calculation
          dudw= -( ZstepB' + W*dzdw);

        %dJdw(k) calculation and keeping the previous derivative of the cost
          prevdJdw=dJdw;
          dJdw=2*ZstepB'*R1cost*dzdw+2*deltaB(k)*R2cost*dudw;

        %dzdw(k+1) calculation
          dzdw=A*dzdw+B*dudw;

        %adaptive learning rate   - to improve convergence speed and accuracy

          if dJdw/prevdJdw<1      % Cost ratio
             gama=1.05*gama;
          end
          if dJdw/prevdJdw>1.005
             gama=0.7*gama;
          end
          deltaw=-gama*dJdw;     % value for deltaw
          gama_next(k+1) = gama;

        %weighting update
          W=W+deltaw;  % Incremental training
           net.IW{1,1} = W;


  %%%%%%%%%%%%%%%%%% END OF WEGHTINGS UPDATE%%
  %%%%%%%%%%%%%%
     %lateral_acceleration calculation
       lateral_accelerationA(k+1) = (ZstepA(2,1)-YdotA)/T+u*ZstepA(4,1);
       lateral_accelerationB(k+1) = (ZstepB(2,1)-YdotB)/T+u*ZstepB(4,1);

     %update absolute positions
       global_positionA(k+1) = global_positionA(k) + u*T*phiA(k) + ZstepA(1,1);
       global_positionB(k+1) = global_positionB(k) + u*T*phiB(k) + ZstepB(1,1);
       phiA(k+1) = phiA(k) + ZstepA(3,1);
       phiB(k+1) = phiB(k) + ZstepB(3,1);
  %store the state
   ZA(:,k+1) = ZstepA;
   ZB(:,k+1) = ZstepB;
end

if epoch ==1
plot_plot
end
toc

gama_last = gama_next(K-n);
gama_end(epoch) = gama_last;
W_last = W;
W_end(epoch,1:4+n+1) = W_last(1,1:4+n+1);
end

Ws = W_init+0.0001;
xyz=[];
for r = 1:jg;
  % xyz(r) =(W_lastB(r)-W_initB(r));
  xyz(r) =abs((net.IW{1,1}(r)-Ws(r))/Ws(r))*100;
end
weight_change=(sum(xyz))/jg;

%%%%%%%%%%%%%%%%%%% END OF MAIN LOOP
%%%%%%%%%%%%%%%%%%

figure(2)
plot(gg)
xlabel('No. of Epoch');
ylabel('Learning Rate');
title('Plot of Learning Rate')
grid on

figure(3)
ww1=W_end(:,10)'
ww = [W_init(10) ww1];
figure(3)
plot(ww)
xlabel('No. of Epoch');
ylabel('Weight');
title('Plot of Updated Weight vs No. of Epoch')
grid on

Plottings2inoneshot

%%%%%%%%%%%%%%%%%%%%END OF LinearNN.m
%%%%%%%%%%%%%%%%%%
```

APPENDIX 2: MATLAB CODE

Neural Control of Nonlinear Car Model (Batch Training)

```
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %              NonLinearCarNN.m           %
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('   -----------------------------------------------------------------')
disp('       NN with 1 neuron, NON Linear Car Model  ')
disp('   -----------------------------------------------------------------')
disp('');
clear all;close all;

% forward speed
   u=input('which speed ?  (using 20 par (default))');
   if isempty(u), u=20; disp('Using u=20m/s (default)'), end

% sampling period T
   T=0.05;

% number of preview points
   n=input('how many preview points (using 20 par (default))');
   if isempty(n), n=2/T; disp('Using a number corresponding to 2sec ahead (default)'), end

%car parameters definition
Cf=0.8*282240;
Cr=0.8*188160;
a=0.92;
b=1.38;
M=1400;
G=17;
Iz=3040;

%Magic Formula Parameters
bm=17.5; %magic formula stiffness parameter, one tyre
cm=1.68; %magic formula shape parameter, one tyre
dmf=0.8*4800; %=3840
dmr=0.8*3200; %=2560
em=0.6;  %magic formula curvature parameter, one tyre

%%%%%%%%%%%%%%%%%%%%% Road Model Matrices
%%%%%%%%%%%%%%%%%%%%%
D=[zeros(n,1) eye(n);
   zeros(1,n+1)];
E=[zeros(n,1); 1];

%%%%%%%%%%%%%%%%%%%%%Linear Car Model
%%%%%%%%%%%%%%%%%%%%%
Linear_car_model
%disp('')
```

```
%%%%%%%%%%%%%%%%%% Linear control gain calculaton
%%%%%%%%%%%%%%%%
%cost prioritites
Q=[100 0 ;
   0  1 ];
R2=1;

%compute the LQG gain Kt
LQRgain

%%%%%%%%%%%%%%%%%%%%%% Linear cost parameters
%%%%%%%%%%%%%%%%
%the cost to be minimised is the folowing one :
%J=Z(:,k)'*R1cost*Z(:,k)+delta(k)'*R2cost*delta(k)

R1cost=R1;
R2cost=R2;

tic
disp('          Loading path information.....')

%%%%%%%%%%%%%%%%%%%%% Path information
%%%%%%%%%%%%%%%%%%%%%%%%%
circuits_2
[K,nb] = size(yref)    % Array size for yref

%%%%%%%%%%%%%% %%%% State definition & initialisation
%%%%%%%%%%%%%%%

%The augmented state comprises the states of the car and the states of the road model
% The notations A and B represents the optimal controller and
% the single processing element  respectively.

ZA = zeros(4+n+1,K-n-1);
ZA(1,1) = yref(1);
ZA(3,1) = (yref(2) - yref(1))/(u*T);
ZA(4+1:4+n+1,1) = yref(1:n+1)';

ZB = zeros(4+n+1,K-n-1);
ZB(1,1) = yref(1);
ZB(3,1) = (yref(2) - yref(1))/(u*T);
ZB(4+1:4+n+1,1) = yref(1:n+1)';

%augmented E matrix
Ebis=[zeros(4,1); E];
%%%%%%%%%%%%%%%%%%%% Paramaters Initialisation
%%%%%%%%%%%%%%%%%%%%
      dzdwB = zeros(n+5,n+5);
      dudwB = zeros(1,n+5);
      dJdwB = zeros(1,n+5);
      prevdJdwB = zeros(1,n+5);
      deltawB = zeros(1,n+5);
      prevdeltawB = zeros(1,n+5);
```

```
%other parameters
   phiA(1)=(yref(2)-yref(1))/(u*T);
   phiB(1)=(yref(2)-yref(1))/(u*T);

   deltaA(1)=0;
   deltaB(1)=0;

   lateral_accelerationA(1)=0;
   lateral_accelerationB(1)=0;

   global_positionA(1)=ZA(1,1);
   global_positionB(1)=ZB(1,1);

   ZinitA = zeros(4+n+1,1);
   ZinitB = zeros(4+n+1,1);

   ZstepA = zeros(4+n+1, 1);
   ZstepB = zeros(4+n+1, 1);


%%%%%%%%%%%%%%%%partial derivative of the side forces
%%%%%%%%%%%%%%%%
  syms var1 % symbolic steer angle
  syms var2 % symbolic local speed
  syms var3 % symbolic rate

  %slip angles
   alphaf=var1/G-atan((var2+a*var3)/abs(u));
   alphar=-atan((var2-b*var3)/abs(u));
  %front and rear lateral forces
   Fyf=2*dmf*sin(cm*atan(bm*alphaf-em*(bm*alphaf-atan(bm*alphaf))));
   Fyr=2*dmr*sin(cm*atan(bm*alphar-em*(bm*alphar-atan(bm*alphar))));
  %partial derivatives
   dFyfdu=diff(Fyf,var1);
   dFyfdx2=diff(Fyf,var2);
   dFyfdx4=diff(Fyf,var3);
   dFyrdx2=diff(Fyr,var2);
   dFyrdx4=diff(Fyr,var3);

%%%%%%%%%% Neural network implementation: single processing element
%%%%%%%%%%

disp('          neural network implementation.....')

%choose an input layer with n+5 (number of states) neurons
inputB=[-50*ones(n+5,1) 50*ones(n+5,1)];

net=newff(inputB,1,{'tansig'},'trainbfg');
net.trainParam.searchFcn = 'srchbac';
net.trainParam.lr = 0.008;
net.trainParam.epochs = 20;
net.trainParam.show = 1;
net.trainParam.goal = 1e-10;
    %initialize the vector W(:) containing all weights and biases.
        for jg=1:4+n+1
        WB(jg)=Kt(jg);
        W_init=WB;
        end

    %initialize neural network weightings
      net.IW{1,1}=WB;
      net.b{1} =[0];
toc
disp('          main loop.....')
tic

                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %                    MAIN LOOP              %
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for k=1:(K-n-1)
    %Weight_nextB(k,4+n+1)=WB(1,4+n+1);

     % definition of a new global frame based on local x and y-axes
     % definition of the states of the car

       ZinitA = ZA(:,k);
       YdotA = ZA(2,k);

       ZinitB = ZB(:,k);
       YdotB = ZB(2,k);

         if k>1
           ZinitA(2) = ZinitA(2)-u*sin((phiA(k)-phiA(k-1)));   %the local y-axis changed
           ZinitB(2) = ZinitB(2)-u*sin((phiB(k)-phiB(k-1)));
         else
           ZinitA(2)= 0;
           ZinitB(2)= 0;
         end

      % due to the choice of the frame, absolute positions become zero
        ZinitA(1) = 0;
        ZinitA(3) = 0;
        ZinitB(1) = 0;
        ZinitB(3) = 0;

      % absolute to relative road data transformation
        local_yrefs = yref(k:k+n+1);
          for j = 1:(n+2),
            local_yrefsA(j) =  local_yrefs(j) - global_positionA(k)- ...
            (j-1)*phiA(k)*u*T;
            local_yrefsB(j) =  local_yrefs(j) - global_positionB(k)- ...
            (j-1)*phiB(k)*u*T;
        end
    % definition of the remaining states (preview path errors)
        ZinitA(4+1:4+n+1) = local_yrefsA(1:n+1);
        ZinitB(4+1:4+n+1) = local_yrefsB(1:n+1);

%%%%%%%%%%%%%%%%%%%% state error
%%%%%%%%%%%%%%%%%%%%
    epsA=ZinitA;
    epsB=ZinitB;

%%%%%%%%%%%%%%%%%%%% steer angle %%%%%%%%%%%%%%%%%%%%
    deltaA(k) = -Kt*epsA;
    deltaB(k) = sim(net,-epsB);

                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                 %     Weighting update single processing element      %
                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        net.IW{1,1} = WB;

          %dF3du calculation
          dF3duB=[0;
             (T/M)*subs(dFyfdu, {var1,var2,var3}, {deltaB(k),ZinitB(2),ZinitB(4)}) ;
              0;
             (T/Iz)*a*subs(dFyfdu, {var1,var2,var3}, {deltaB(k),ZinitB(2),ZinitB(4)}) ;
              zeros(n+1,1)];

         %dF3dz calculation

 F3B=  [1    T                                    0    0;

     0    1+(T/M)*(subs(dFyfdx2, {var1,var2,var3}, {deltaB(k),ZinitB(2),ZinitB(4)}) ...
             +   subs(dFyrdx2, {var2,var3}, {ZinitB(2),ZinitB(4)}) )          0    (T/M)*(
 subs(dFyrdx4, {var2,var3}, {ZinitB(2),ZinitB(4)}) ...
                                                         + subs(dFyfdx4, {var1,var2,var3},
 {deltaB(k),ZinitB(2),ZinitB(4)}) )    ;

     0    0                                   1    T;


     0    (T/Iz)*( a*subs(dFyfdx2, {var1,var2,var3}, {deltaB(k),ZinitB(2),ZinitB(4)}) ...
             -b* subs(dFyrdx2, {var2,var3}, {ZinitB(2),ZinitB(4)}) )           0
 1+(T/Iz)*( a*subs(dFyfdx4, {var1,var2,var3}, {deltaB(k),ZinitB(2),ZinitB(4)}) ...
                                                           -b* subs(dFyrdx4, {var2,var3},
 {ZinitB(2),ZinitB(4)}) ) ] ;


    dF3dzB=[F3B       zeros(4,n+1); zeros(n+1,4)     D  ];
     dzdwB=dF3dzB*dzdwB+dF3duB*dudwB;

 %%%%%%%%%%%%%% END OF WEGHTINGS UPDATE
 %%%%%%%%%%%%%%
 %side forces
 %F1=2*dm*sin(cm*atan(bm*alpha-em*(bm*alpha-atan(bm*alpha))));

    front_slip_angleA=deltaA(k)/G-atan((ZinitA(2)+a*ZinitA(4))/u);
    Z13A=bm*front_slip_angleA-atan(bm*front_slip_angleA);

    FyfrontA=2*dmf*sin(cm*atan(bm*front_slip_angleA-em*Z13A));

    rear_slip_angleA=-atan((ZinitA(2)-b*ZinitA(4))/u);
    Z25A=bm*rear_slip_angleA-atan(bm*rear_slip_angleA);

    FyrearA=2*dmr*sin(cm*atan(bm*rear_slip_angleA-em*Z25A));


    front_slip_angleB=deltaB(k)/G-atan((ZinitB(2)+a*ZinitB(4))/u);
    Z13B=bm*front_slip_angleB-atan(bm*front_slip_angleB);

    FyfrontB=2*dmf*sin(cm*atan(bm*front_slip_angleB-em*Z13B));

    rear_slip_angleB=-atan((ZinitB(2)-b*ZinitB(4))/u);
    Z25B=bm*rear_slip_angleB-atan(bm*rear_slip_angleB);

    FyrearB=2*dmr*sin(cm*atan(bm*rear_slip_angleB-em*Z25B));

 %acceleration equations
 %differential_equation & state update
    ZstepA(1)=ZinitA(1)+T*ZinitA(2);
    ZstepA(2)=ZinitA(2)+T*((1/M)*(FyfrontA+FyrearA));
    ZstepA(3)=ZinitA(3)+T*ZinitA(4);
    ZstepA(4)=ZinitA(4)+T*(1/Iz)*(a*FyfrontA-b*FyrearA);

    ZstepB(1)=ZinitB(1)+T*ZinitB(2);
    ZstepB(2)=ZinitB(2)+T*((1/M)*(FyfrontB+FyrearB));
    ZstepB(3)=ZinitB(3)+T*ZinitB(4);
    ZstepB(4)=ZinitB(4)+T*(1/Iz)*(a*FyfrontB-b*FyrearB);

 %lateral_acceleration calculation
    lateral_accelerationA(k+1) = (ZstepA(2,1)-YdotA)/T+u*ZstepA(4,1);
    lateral_accelerationB(k+1) = (ZstepB(2,1)-YdotB)/T+u*ZstepB(4,1);

 %update absolute positions
    global_positionA(k+1) = global_positionA(k) + u*T*phiA(k) + ZstepA(1,1);
    global_positionB(k+1) = global_positionB(k) + u*T*phiB(k) + ZstepB(1,1);

    phiA(k+1) = phiA(k) + ZstepA(3,1);
    phiB(k+1) = phiB(k) + ZstepB(3,1);

 %store the state
  ZA(:,k+1) = ZstepA;
  ZB(:,k+1) = ZstepB;
end
```

```
net = train(net,-epsB);
toc

Ws = W_init+0.00001;
xyz=[];
for r = 1:jg;
  xyz(r) =abs((net.IW{1,1}(r)-Ws(r))/Ws(r))*100;
end
weight_change=sum(xyz)/jg;

%%%%%%%%%%%%%%%%%%% END OF MAIN LOOP
%%%%%%%%%%%%%%%%%%%

Plottings2inoneshot

%%%%%%%%%%%%%%%%%%%END OF NonLinear Car NN
%%%%%%%%%%%%%%%
```