CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Advances in Fuzzy Logic Control for Lateral Vehicle Guidance

Thomas Hessburg
Masayoshi Tomizuka

California PATH Working Paper
UCB-ITS-PWP-94-04

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department Transportation, Federal Highway Administration.

March 1994

ISSN 1055-1417

Advances in Fuzzy Logic Control for Lateral Vehicle Guidance

Thomas Hessburg and Masayoshi Tomizuka

PATH Research Report, MOU 89

Fiscal year 1993/94

Abstract

A candidate for intelligent control for lateral guidance of a vehicle is a fuzzy logic controller (FLC). The details of the most recent FLC with a rule base derived from heuristics and designer insight to the problem at hand, as well as a method for estimating a projected lateral displacement are presented along with simulation results. The structure of this FLC is suited to incorporate human knowledge about the steering operation by its choice of inputs and outputs of the FLC which are natural to human steering operation. In addition, a method that makes use of preview information regarding upcoming road curvature is developed and simulated based on human steering operation. This method projects an estimate of the lateral error relative to the reference track at a specified look-ahead time. Simulations imply that there exists an optimal look-ahead time. Too short a time does not make full use of the preview information, while too long a time results in inaccurate estimates of future lateral error, degrading the performance. Success of the FLC strongly depends on the set of rules provided by the designer. It is extremely important to have methodologies for rule development. Therefore, a methodology is considered for automatically generating a fuzzy rule base controller, specifically, a Neural-Network Driven Fuzzy Logic Controller (NNDFLC). This process automates the design of control parameters and uses a clustering technique to cluster the space of inputs to the controller in order to take advantage of local characteristics in the reasoning of the control outputs. Given enough input/output data to train the neural networks involved, simulations show its success. However, this method requires the generation of a complete set of training data, covering the entire space of inputs to the controller. This is an important factor in the design of the NNDFLC with regard to the convergence of control parameters in the learning process and for taking advantage of the clustering features of the controller.

Table of Contents

1 Introduction

An important component in Intelligent Vehicle Highway Systems (IVHS) is the lateral motion control of a vehicle in lane following maneuvers as well as lane change maneuvers. This report focuses on lane following, where the objective is to track the center of a lane with smooth ride quality under a range of operating conditions such as vehicle speed, wind gusts, and road conditions. The reference/sensing system consists of discrete magnetic markers embedded in the roadway, forming a predetermined path [6, 15]. Magnetometers are mounted on the front of the vehicle. **A** computer algorithm computes the lateral displacement from the horizontal and vertical components of the magnetic field signal.

The present effort complements the study on investigating the use of a FLC to control the lateral motion of a vehicle [5]. There are many features of lateral guidance of a vehicle which motivate the use of a FLC. They include the substantial nonlinearities found in vehicle dynamics, especially in the tire characteristics. Also, information is available regarding human expertise on steering maneuvers of a vehicle. In addition, a FLC has great flexibility on the use of inputs and outputs. The results of the FLC can be compared to a linear lateral guidance controller using the frequency shaped linear quadratic (FSLQ) and preview theory, which has been developed and successfully implemented on a Toyota Celica [12].

A FLC for vehicle lateral control was proposed in [5]. The rule base was developed based on engineering judgment. The parameters of the FLC, which define membership functions and the consequent expressions, were tuned manually. The theme of this FLC was to formulate inference rules based on PID (proportional, integral, and derivative) control techniques. Therefore, the inputs to the FLC were chosen to be the lateral error between a sensor mounted on the vehicle and the center of the road, y, change in the lateral error between measurement samples, *cy*, and the summation of the lateral error taken at each measurement sample, *Cy*. The output of the FLC was the absolute command for the front wheel steering angle, 6,. In addition, preview information regarding road curvature was used as input for generating feedforward control action. This feedforward control term is based on a steady state estimate of front wheel steering angle derived from an expression obtained in [11]. This steering term was gradually implemented at curve transitions based on human observations obtained in [4] as well as observations of the preview steering term developed using preview control theory [12]. Specifically, the implementation of the preview steering angle was a ramp from zero (at a prescribed time *before* the curve transition) to the calculated steady state preview steering angle (at a prescribed time *after* the curve transition). Although the simulation results showed good performance, the rule base was somewhat unnatural from the viewpoint of human expertise. For example, inferring the *absolute* front wheel steering angle, δ_c, from y, the time derivative of y, and the time integral of y, is not natural for human drivers.

A different approach to developing a FLC based on engineering judgment is proposed in section *2* of this report. The theme of this FLC is to use rules which are more natural to human driving maneuvers. The inputs to this FLC are y, $\dot{y}$ and the measured yaw rate relative to the desired yaw rate, $(\dot{\varepsilon} - \dot{\varepsilon}_d)$, where $\dot{\varepsilon}_d$ depends on the velocity, v, of the vehicle and the radius of curvature, ρ, of the reference track such that $\dot{\varepsilon}_d = v/\rho$. The output is the rate of change in the command to the front wheel steering angle, $\dot{\delta}_c$. In addition, an alternative method of incorporating the road curvature preview information is developed.

Another issue that needs to be addressed is the automation of tuning the parameters that define a FLC. This is a very active topic in current FLC research. Manual tuning may be inefficient when considering the number of different vehicles involved in an automated highway. Some methods developed include gradient descent [1, 10], neural networks [8, 14], and genetic algorithms [9]. In section 3 we investigate a neural network approach to automate membership function generation and consequent expressions to form a FLC with a rule base which has a trade-off between automation and an intuitive feel. This approach is similar in structure to the ellipsoidal learning technique developed by Dickerson and Kosko [2, 3], used in longitudinal control of vehicles.

2 Fuzzy Logic Controller (Manual Tuning)

The basic fuzzy algorithm detailed in [5] is used to develop a FLC based on inference rules with linguistic variables *LE* (lateral error), *CLE* (change in lateral error with respect to time), and *YWR* (yaw rate relative to desired yaw rate) in the antecedents and *CA* (change in front wheel steering angle command with respect to time) in the consequents. The choice of linguistic variables is based on human steering intuition. For example, given a state of the vehicle relative to the current road geometry, a human driver reasons how much to *change* the front wheel steering angle as opposed to reasoning what the *absolute* front wheel steering angle should be. Also, lateral error is critical for reasoning how to track a lane, while the change in lateral error and yaw rate of the vehicle are used to reason how to realize a comfortable ride.

The rule base is made up of rules taking the form:

IF LE is A, AND CLE is A, AND YWR is A_{YWR} THEN CA is $A_{C\Delta}$

where A_i represents linguistic values of linguistic variable, i.

$$\begin{aligned}
A_{LE} &\in \{NB, NS, NIL, PS, PB\},\\
A_{CLE} &\in \{NB, NS, NIL, PS, PB\},\\
A_{YWR} &\in \{NEG, NIL, POS\},\\
A_{C\Delta} &\in \{NH, NB, NM, NS, NIL, PS, PM, PB, PH\}
\end{aligned}$$

(for example: $NB \equiv$ Negative Big and $NH \equiv$ Negative Huge)

The linguistic values of the antecedents are defined precisely by membership functions, after manual tuning, in Appendix **A.** Also, the consequent parts of the rules are defined by singletons (see Appendix A). Note that the number of fuzzy subsets of *LE* (i.e. the number of different linguistic values of *LE)* is 5. That of *CLE* and *YWR* are 5 and 3, respectively. Therefore, there are 5x5x3 = 75 rules in the rule base.

The block diagram of the closed loop system is seen in Fig. 2.1. The control law is executed at a constant time sample, t_s. Therefore, y and **y** are assutned to be available as estimates, and $\dot{\varepsilon}$ is measurable at any time, including the time in between markers.

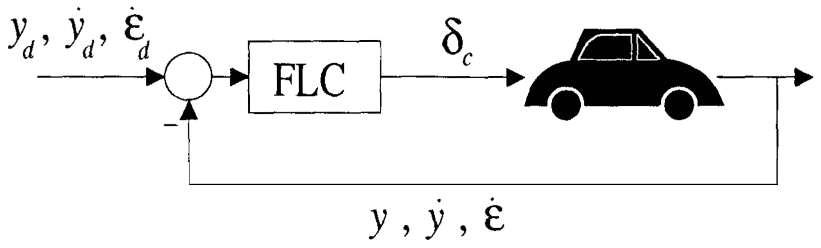
y_d, $\dot{y}_d$, $\dot{\varepsilon}_d$
FLC
δ_c
−
y , $\dot{y}$, $\dot{\varepsilon}$

Figure 2.1: Closed Loop Block Diagram

Fig. 2.2 shows a simulation, using a nonlinear model [11], of a straight section of roadway where the vehicle started with an initial condition of 0.3 meters of lateral displacement. Conditions are nominal as stated in [5]. The closed loop block diagram becomes a regulation system in a straight roadway as the quantities y_d, $\dot{y}_d$, and $\dot{\varepsilon}_d$ are all zero.

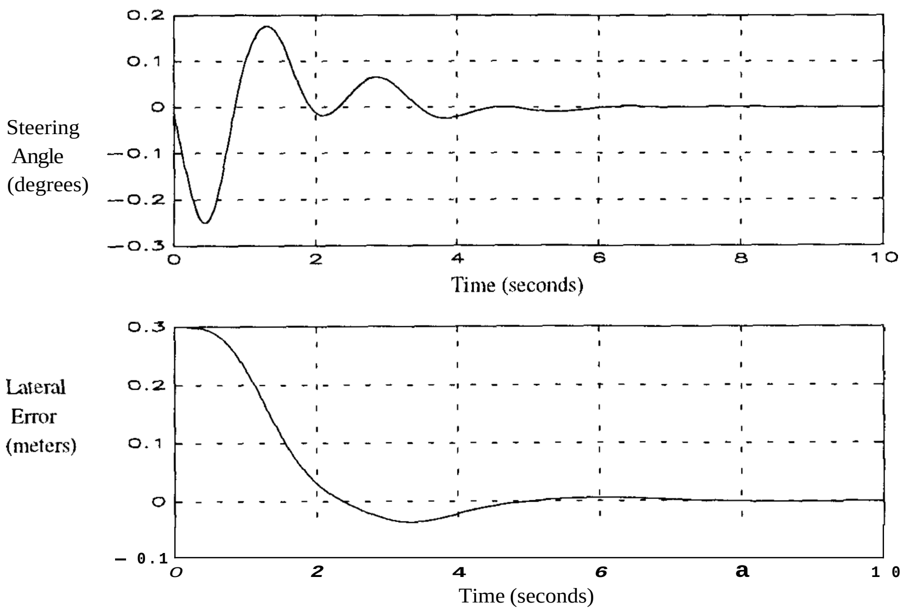

Figure 2.2: Straight Roadway Simulation; Nominal Conditions

The steering command, δ_c, exhibits slightly more oscillation as compared to the FSLQ controller [11]. This can be attributed to the fact that the FLC infers $\dot{\delta}_c$, as opposed to δ_c, which has the effect of adding an order to the closed loop system. In addition, this FLC has a slower response in comparison to the FSLQ controller in terms of rise time, t,, which we define to be the time required for the lateral displacement of the closed loop system to go from 0.3 meters to 0.03 meters. For the FSLQ, $t_r = 0.9$ seconds, and for this FLC, $t_r = 2.0$ seconds.

Next we consider how to utilize road curvature preview information. The preview information provided is the radius of curvature on the roadway and the location of the starting and ending points of the curvature. The idea is to estimate the lateral position of the vehicle at a time, *t,,* in the future. This estimate is then subtracted from an estimated desired lateral position derived from the preview information. The result is used as the lateral error in the FLC rule base.

This preview method is explained in detail by referring to Fig. 2.3. For the derivation, t_c is denoted as the current time. The controller has access to road curvature information for $t \in [t_c \ (t_c + t,)]$. The first step is to estimate $\hat{y}_{t_l}(t_c)$, the lateral error of the vehicle at time t_l after the current time with respect to a line tangent to the line of the reference path taken at its current position (see Fig. 2.3).

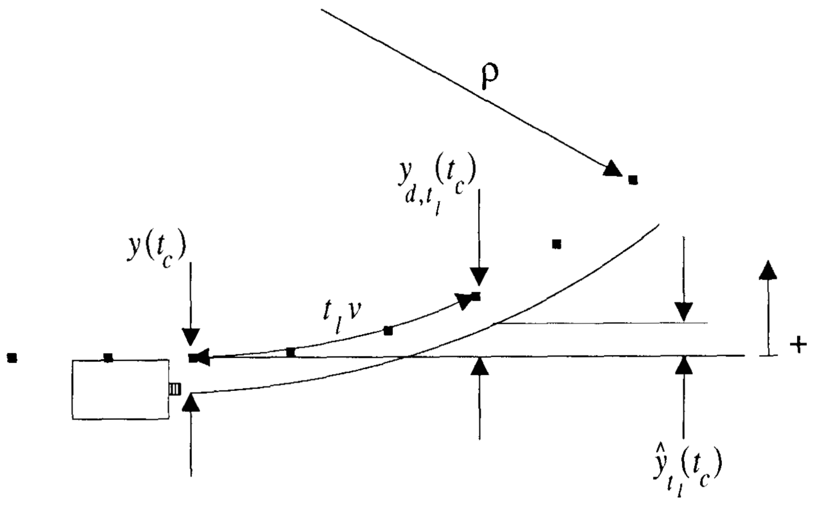

Figure 2.3: Schematic of Preview Method

This estimation uses the current state of the vehicle, $y(t_c)$, $\dot{y}(t_c)$, and $\dot{\varepsilon}(t_c)$. **A** vehicle with a constant yaw rate, $\dot{\varepsilon}$, will travel at a radius of curvature of $v/\dot{\varepsilon}$, where v is the longitudinal velocity. Thus, the lateral acceleration, v^2/ρ, is approximately $v\dot{\varepsilon}$. Integrating the acceleration twice with respect to time over a time interval $[t_c \;\; (t_c + t_l)]$, we get

$$\hat{y}_{t_l}(t_c) = y(t_c) + \dot{y}(t_c)t_l + v\dot{\varepsilon}(t_c)\frac{t_l^2}{2} \tag{2.1}$$

The second step is to estimate the desired lateral error, $y_{d,t_l}(t_c)$, of the vehicle at time t_l after the current time with respect to a line tangent to the reference path taken at its current position. This estimate, $y_{d,t_l}(t_c)$, is a function of the radius of curvature of the roadway, ρ, provided by preview information. Suppose the vehicle is at the start of the curvature at time t_c. The travel distance ahead is $t_l v$. Therefore, given ρ, the following expression can be obtained:

$$y_{d,t_l}(t_c) = \rho\left\{1 - \cos\left(\frac{t_l v}{\rho}\right)\right\} \tag{2.2}$$

Approximating $\cos(x)$ by $(1 - x^2/2!)$ (using the first two terms of the Taylor's series expansion and assuming $t_l v << \rho$), we get

$$y_{d,t_l}(t_c) \approx \frac{(t_l v)^2}{2\rho} \tag{2.3}$$

In general the formulation of $y_{d,t_l}(t_c)$ should be derived when t_c does not occur at the beginning of the curve. For simplification, we assume that the radius of curvature, ρ, remains constant over every curved section. This implies that our preview time window, t_l, contains at most two radii of curvature, ρ_c and ρ_n, the current and next radii of curvature, respectively (see Fig. 2.4).

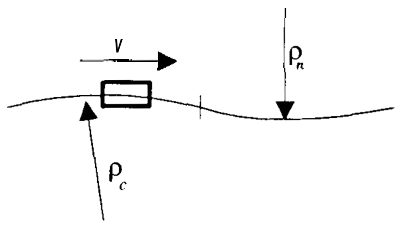

Figure 2.4: Preview Concept

If we define a term, t_2, as the time at which the vehicle will reach the next curve transition or t_l, whichever is smaller, then we can generalize Eqn. (2.3) as

$$y_{d,t_l}(t_c) \approx \frac{(t_2 v)^2}{2\rho_c} + \frac{((t_l - t_2)v)^2}{2\rho_n} \tag{2.4}$$

The quantities $(\hat{y}_{t_l}(t_c) - y_{d,t_l}(t_c))$ and $(\dot{\varepsilon} - \dot{\varepsilon}_d)$, where $\dot{\varepsilon}_d = v/\rho_c$, are used as the base variables for *LE* and *YWR* membership functions, respectively. Figure 2.5 shows a simulation of curved roadway with a radius of curvature of 300 meters and a vehicle speed of 20 (meters/second). The look-ahead time, t_l, is 0.5 seconds, which yielded the best results in simulation. The road is straight for the first ***2*** seconds, curved for the next 5 seconds, and ends off straight for the remaining time.

The largest deviation from the center of the roadway is 5 centimeters. **As** the vehicle approaches the beginning of the curve, the vehicle begins to turn inside the curve to nearly 1 centimeter, and then the vehicle slides outside the curve about 5 Centimeters.

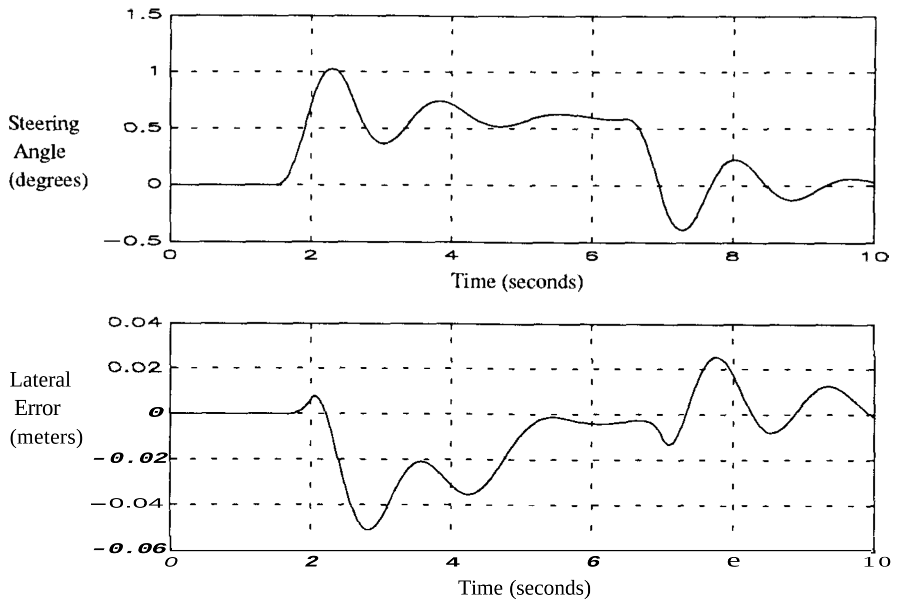

Figure 2.5: Curved Roadway Simulation, t_l = 0.5 seconds; Nominal Conditions

Figure 2.6 shows the effect of a longer preview time , t_l = 1.0 seconds. At the response time of 1 second the vehicle starts to steer inside the curve, 0.5 seconds earlier than the simulation of Fig. 2.5. Unlike the smooth steering action from the straight roadway to the steady state steering angle of the curved section seen in Fig. 2.5, the simulation with t_l = 1.0 seconds has an oscillation. In addition, the case of t_l = 1.0 seconds has an initial lateral error of 15 Centimeters and takes nearly **3** seconds to reduce the lateral error to 2 centimeters.

The existence of the optimal preview time length in the present FLC scheme is contrasted to the optimal preview theory in Peng [12], which suggests that the performance is improved as the preview time length is increased. In the present FLC scheme the predicted error is based on the extrapolation of the lateral position, lateral velocity, and yaw rate (see Eqn. (2.1)) over the preview time span. In fact the lateral velocity and yaw rate are not fixed quantities over the preview time span. Thus, this estimation scheme becomes more inaccurate if the preview length is increased. In the optimal preview theory time variations of the lateral velocity and yaw rate are taken into consideration.

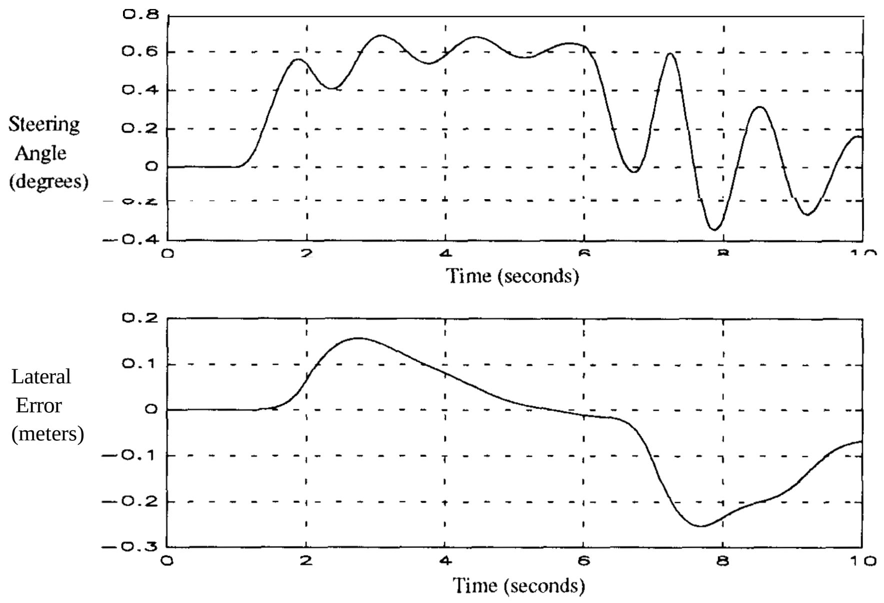

Figure 2.6: Curved Roadway Simulation, t_l = 1.0 seconds; Nominal Conditions

Figure **2.7** shows the effect of a shorter preview time , $t_l = 0.3$ seconds. In comparison to the simulation of $t_l = 0.5$ seconds (Fig. 2.5), the simulation with the shorter preview length steers the vehicle less inside the curve (less than 1 centimeter), but slides further outside the curve to nearly 15 centimeters.

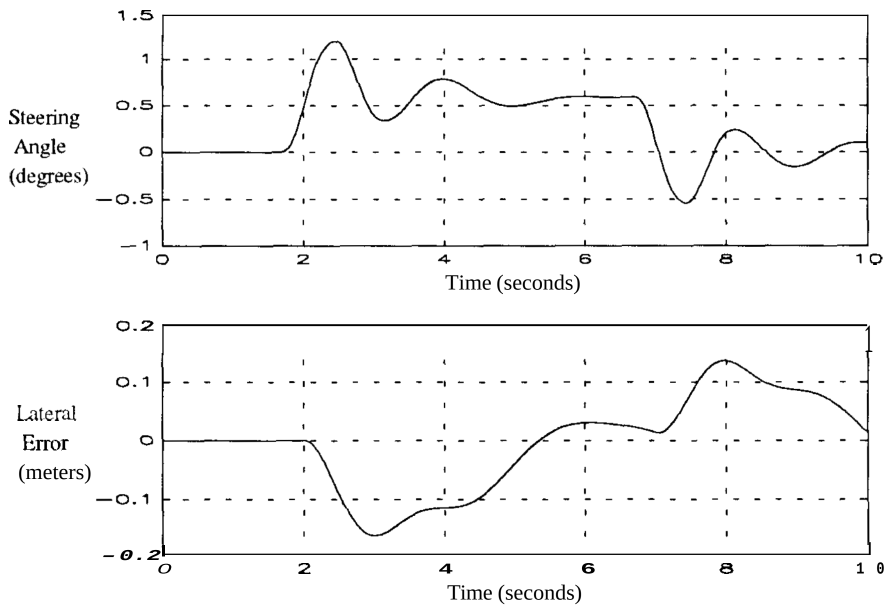

Figure 2.7: Curved Roadway Simulation, t_l = 0.3 seconds; Nominal Conditions

In the case of using feedback control without preview information or with preview lengths less than $t_l = 0.3$ seconds, this FLC could not track the curved section, as the time response of the lateral displacement exhibited unstable oscillations of increasing amplitude.

3 Neural Network Driven Fuzzy Logic Controller

The Neural-Network Driven Fuzzy Logic Controller (NNDFLC) method of fuzzy rule development addresses several issues that plague the design of a manually tuned fuzzy logic controller. These include the lack of a systematic way of generating membership functions. The NNDFLC is capable of generating general, highly nonlinear membership functions. In addition, the NNDFLC addresses the issue of incorporating a method of learning into the control system, a desirable feature of neural-networks. The NNDFLC is essentially a rule base controller that has key elements enhanced by utilizing neural-networks. However, a drawback of the NNDFLC is the requirement of training data for the neural networks.

To begin with we use a single neural network, NN_μ, to provide a way of automating the design of general, highly nonlinear membership functions. NN, is a function map from the inputs to the controller to the degrees of membership that these inputs have to each rule. This is in contrast to the conventional design approach which uses engineering judgment to denote a membership function with two or three parameters such as a triangle or trapezoid, and whose membership functions are based on one input variable at a time. In NNDFLC we assign values to membership functions based on the forward propagation of NN,. In addition, the membership functions may depend on more than one input variable as will be observed in the following discussion. **Also,** being a neural network, NN_μ has the capability of learning from desired training data and realizing highly nonlinear in calculating membership functions.

Several methods have been proposed for the specification of the consequent part of fuzzy rules, for example using fuzzy subsets, using singletons, and using a linear combination of the input variables. We propose incorporating neural networks in the phase of consequent evaluation, as well. These neural networks are denoted as NN_c^s, where $s = 1. 2. \dots n_r$, and n_r is the total number of rules.

We want to show the feasibility of this method (i.e. the capability of tuning the NNDFLC) by generating training data (TRD) from an existing controller. We begin with a simple case of a PD controller. Fixing the velocity of the vehicle, a proportional and derivative (PD) controller was designed such that the closed loop poles of a discretised simplified linear model of the system were inside the unit circle. This PD controller acts as a teacher for the NNDFLC. The TRD for the NNDFLC can be generated by uniformly covering a reasonable range of the space of inputs to the controller made up of elements, $\mathbf{x} = [y\ \dot{y}]^T$. The corresponding δ_t can be determined from the PD control law, Thus, our TRD consists of a 2 dimensional input to the controller, $\mathbf{x} = [y\ \dot{y}]^T$ and a one dimensional output, 6,.

The general idea is to take the set of TRD and partition it into clusters, each one representing an inference rule. This is done by a clustering method performed on only the input of the TRD. The next step is to identify the antecedents (the IF part) which amounts to training NN_μ to infer the n_r membership functions, which measure the degree to which the input vector, $\mathbf{x}$, belongs to each of the clustered regions. The final step is to identify the consequent parts (the THEN part). This is done by training each NN_c^s, for s = 1. **2. ...** n_r.

The general form of the rule base is as follows:

$$\text{IF } \mathbf{X} \text{ is } A, \qquad \text{THEN } \delta_c^s = NN_c^s(\mathbf{x}) \qquad \text{for s} = 1.\ 2.\ \ldots n_r \tag{3.1}$$

where:

$\mathbf{X}$ is the linguistic variable of the vector $\mathbf{x}$

A_s is the linguistic value representing the s-th region in the input space.

δ_c^s is the front wheel steering angle inferred by NN_c^s.

Here, the methodology is detailed step by step.

Step 1: A clustering method is performed on the input, $\mathbf{x}$, of the TRD to create a clustering dendrogram (see Appendix B). We denote the total number of TRD as n,. The best number of partitions of the input set is detennined by judging the distances between clusters in this clustering dendrogram. Once the number of partitions is chosen, we denote each partition as R_s for s = 1. 2. ... n_r. Note that this is a hard cluster so each $\mathbf{x}_i$ in the TRD must belong to one and only one R_s. The number n_r is the number of inference rules that the NNDFLC will have.

Step 2: *NN,* is trained to identify the antecedents of Eqn. (3.1). The number of input cells to NN_μ is equal to the number of input variables, in our case, two. *NN,* is a multilayered neural network and takes the form of Fig. 3.1, in our simple example of using y and $\dot{y}$ **as** inputs to the controller. Note that the "1" values input to nodes in the input and hidden layers are a standard feature of multilayered neural networks to allow the neuron computations to have a bias term.

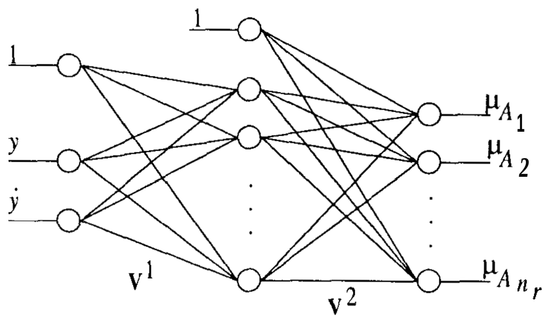

Figure 3.1: Structure of NN_μ

This neural network has an input layer, a hidden layer whose number of cells depends on the complexity needed by the specific problem, and an output layer with the number of cells equal to the number of inference rules, n_r. The values of the outputs are equal to the degree that **x** belongs to A_s for s = 1. 2. ... n,, which we denote as the membership function, $\mu_{A_s}(\mathbf{x})$. The matrices $\mathbf{V}^1$ and $\mathbf{V}^2$ represent weight parameters of the neural network. The role of these weights and the back propagation method of training neural networks [13] is detailed in Appendix C.

Included in the TRD are values for the input to *NN,*, but we must specify the desired output NN_μ. Considering Step 1, we denote the desired output of *NN,* as w_i^s, where

$$w_i^s = \begin{pmatrix} 1; & x_i \in R_s \\ 0; & x_i \notin R_s \end{pmatrix} \qquad \text{for } i = 1, \ldots n_t; \quad s = 1, \ldots n, \tag{3.2}$$

Thus, the final weights of the trained NN_μ are used in the evaluation of the antecedents of the NNDFLC.

Step 3: A total of n_r neural networks (NN_c^s for $s = 1, 2, \ldots n_r$) are trained, identifying the consequents of Eqn. (3.1). The training of the s-th neural network uses only data pairs ($\mathbf{x}_i, \delta_t^i$) such that $\mathbf{x}_i \in R_s$. Like NN_r, the number of input cells to NN_c^s is equal to the number of input variables. Each NN_c^s has a similar structure to NN_r, as seen in Fig. 3.2.

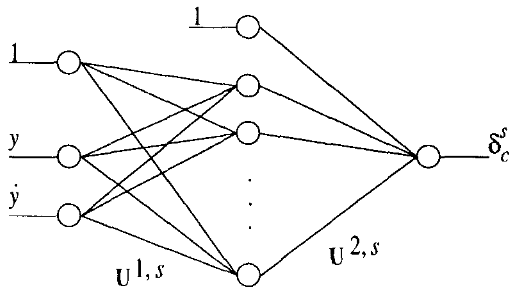

Figure 3.2: Structure of NN_c^s

Note that each NN_c^s has only one output cell, the front wheel steering angle that NN_c^s infers from the input, denoted as δ_c^s. The matrices $\mathbf{U}^{1,s}$ and $\mathbf{U}^{2,s}$ are the weight parameters for NN_c^s.

Let us consider, as a sub-training set of data (STRD$_s$), only those $\mathbf{x}_i$ in the TRD that belong to R_s, and the corresponding desired controller output, δ_t^i. We use STRD$_s$ to train NN_c^s, for $s = 1, 2, \ldots n_r$, to infer the front wheel steering angle.

Step 4: A weighted average method is used for defuzzification. The final command for the front wheel steering angle, δ_c, becomes:

$$\delta_c = \sum_{s=1}^{n_r} \delta_c^s \mu_{A_s} \Big/ \sum_{s=1}^{n_r} \mu_{A_s} \tag{3.3}$$

At this point, the final weights of the trained neural networks can be used to implement the NNDFLC.

When all training is complete, the NNDFLC is implemented according to Fig. 3.3.

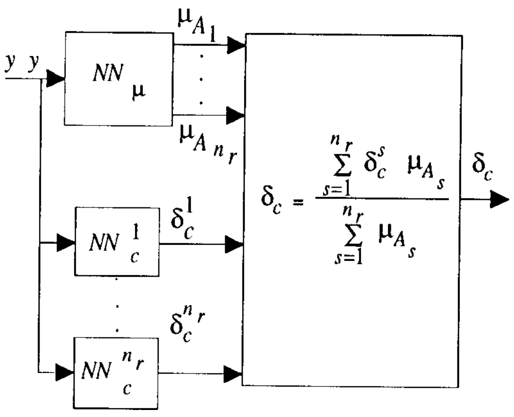

Figure 3.3: Block Diagram of the NNDFLC

Figure **3.4** shows a simulation of a PD controller on a straight roadway. Figure 3.5 shows the results of using training data from the PD control law to generate a NNDFLC, with five clusters.

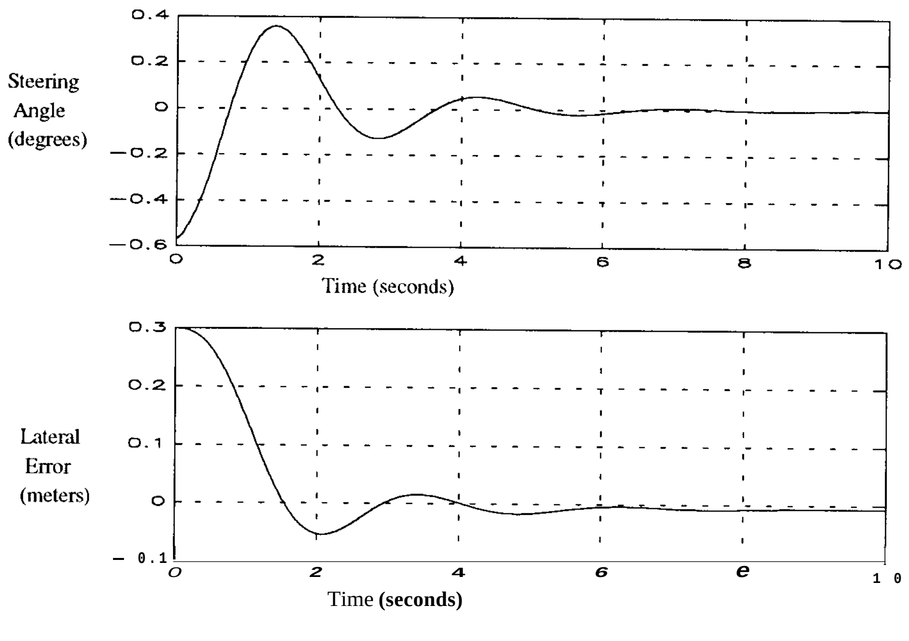

Figure 3.4: PD Control Simulation; Nominal Conditions

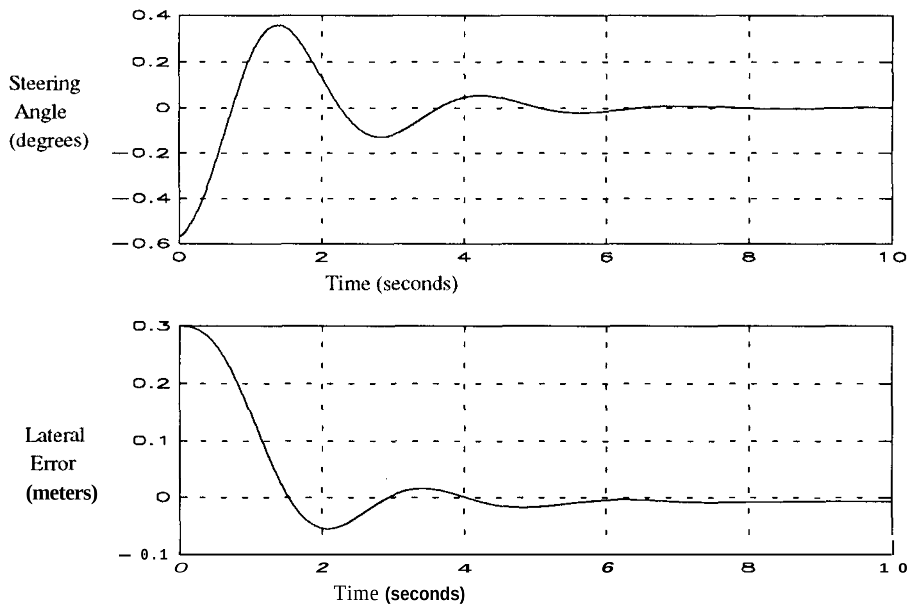

Figure 3.5: NNDFLC Control Simulation; Nominal Conditions

The results of the NNDFLC are nearly identical to that of the PD controller. In fact, a simulation of the NNDFLC with one cluster is also in agreement with the PD controller. This is not surprising since it is not difficult for a neural network to closely approximate a globally linear mapping such as PD control.

A set of training data generated by human drivers was also used to design a NNDFLC. This data consisted of twelve trails of driving on the same track used for lateral control experiments at the Richmond Field Station. This data was highly inconsistent and very non-uniform in the space of inputs to the controller. Thus, the neural networks associated with the NNDFLC had very poor convergence properties leading to the conclusion that human training data must be conditioned somehow in order to properly design a NNDFLC.

4 Conclusions

A new manually tuned fuzzy logic controller (FLC) was developed based on a rule base derived from engineering judgment and heuristics. Preview information was incorporated into the control in a way natural to human driving operation by using road curvature information to compare the estimated projected position of the vehicle and to a desired projected position of the vehicle, and using this information as input to the FLC. The simulation results of this preview scheme appears to show the existence of an optimal preview time span. Some limitations of this manually tuned FLC design method are the time required to tune the control parameters by trial and error and the lack of a method to relate these parameters to some performance expression of the vehicle. An alternative and perhaps more consistent method to incorporate preview action is suggested and successfully implemented in [7] based on preview control theory for a vehicle [12].

An approach using a neural network driven fuzzy logic controller (NNDFLC) was considered to automate the tuning process of the controller. This method requires a set of training data which maps the states of the vehicle to a control action in terms of front wheel steering angle. Data from human vehicle operation is a logical choice of the generation of the training data. However, the inconsistent nature of such data resulted in very poor learning performance of the neural networks associated with the NNDFLC. As a proof of concept, existing controllers were considered to generate training data. However, the global nature of these controllers removed the advantage of the clustering aspect of the NNDFLC. Thus, the method works, but the generation of the training data must be carefully considered in order to take advantage of the clustering characteristics of the NNDFLC. An area of future research will consider the possibility of blending the NNDFLC and a conventional controller. The NNDFLC will be used in the regions of the input space where a sufficient amount of training data is available, while the other regions would use a control law from a conventional control design such as PID. In addition methods other than NNDFLC will be considered to automate the rule base development.

Appendix A: Membership Function and Singleton Definitions for the FLC

This appendix defines precisely the membership functions and singletons used for the manually tuned FLC. The following graphs show the membership functions for the linguistic variables of the antecedents, *LE*, *CLE*, and *YWR*.

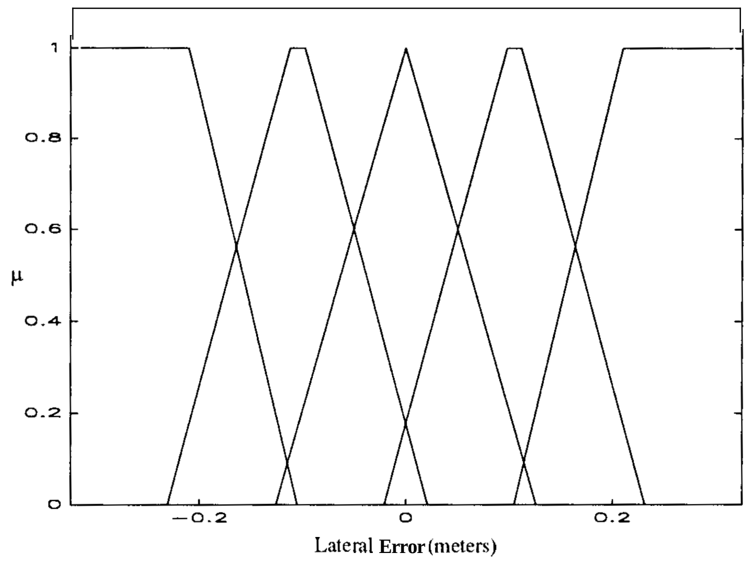

Figure A.l: Membership Functions for *LE*

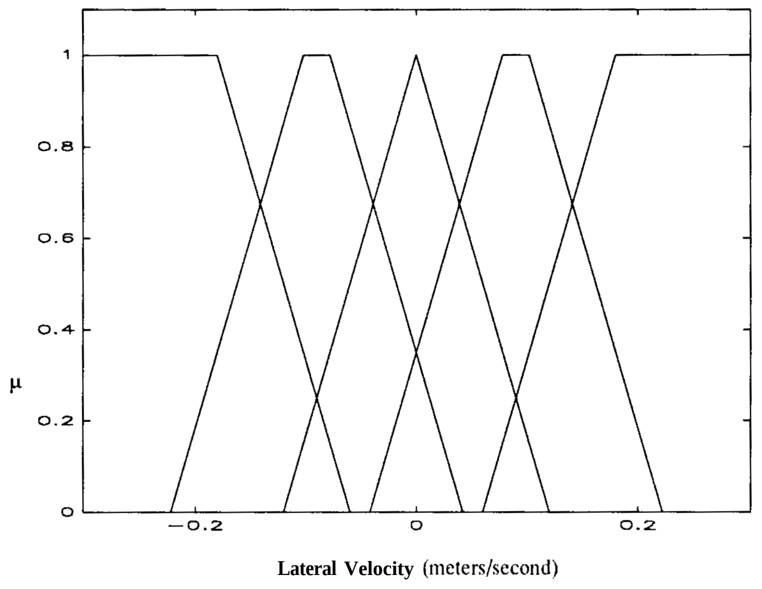

Figure A.2: Membership Functions for *CLE*

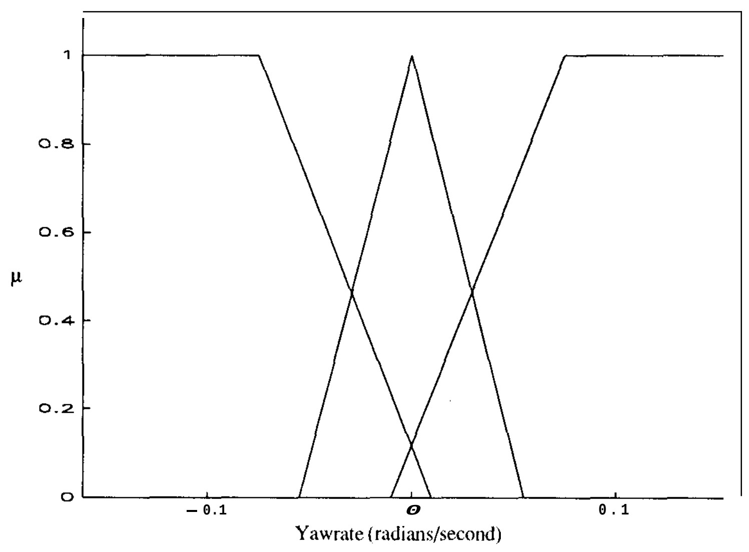

Figure A.3: Membership Functions for *YWR*

The linguistic values associated with the linguistic output, **CA,** are singleton values in radians. They are defined as follows (in radians/second):

NH	=	-0.02
NB	=	-0.015
NM	=	-0.01
NS	=	-0.005
NIL	=	0.0
PS	=	0.005
PM	=	0.01
PB	=	0.015
PH	=	0.02

Appendix B: Data Clustering

This appendix describes the method of clustering data. The purpose is to take a set of n_t data points and group them into reasonable regions. The tasks are 1) to do bookkeeping to note which points should be grouped together as we iterate from n_t clusters to 1 cluster, and 2) to choose the number of clusters which appropriately group the input data. The method is as follows:

1) Create a Cluster Dendrogram
2) Choose Number of Clusters

A cluster dendrogram is a tree structure which displays a distance measure between cluster centers. In our case the distance measure used is the Euclidean norm. To create a cluster dendrogram for n_t data points, we start with n_t clusters with cluster centers equal to the data point values. In the fiist iteration, the two cluster centers with the smallest distance measure between them are grouped to a single cluster and the average of these data points is used as the center of this new cluster. The total number of clusters is now $n_t - 1$. This process is iterated a total of $n_t - 1$ times, keeping track of the distance measures at each iteration. The final iteration will correspond to a single cluster containing all the data points.

The next step is to choose the appropriate number of clusters. This is done by observing the distance measures at each iteration step. For example, if a two dimensional data space has three groups of data (i.e. each data point is close to one of three points in the data space), then the distance measures will be relatively small from the beginning of the iteration steps of creating the cluster dendrogram until the last two distance measures, which will be relatively large. Therefore, the iteration step where a large change in the relative distance measures occurs will provide the logical choice of the number of clusters. In this simple example, the large change in distance measures occurs between iteration step $n_t - 2$ and $n_t - 3$, which means that we should choose **3** clusters.

Appendix C: Neural Network Structure and Training

This appendix details the structure and training of the multilayered neural networks used in the NNDFLC. **A** neural network is a function approximator. Given an input vector, a neural network can approximate the appropriate output vector by the proper choice of weights which parameterize the neural network. In order for a neural network to approximate a specific function, a set of training data is needed. This training data consists of several input/output pairs. To properly adjust the neural network we define a squared error term between the output vector of the neural network and the output vector of the training data for the corresponding input vector of the training data. The training of the neural network is the method of adjusting the weights of the neural network to minimize the squared error term.

We will detail the structure and training of NN_μ and note that NN_c^s will be similar. Figure C.1 shows the multilayered neural network structure of NN_μ.

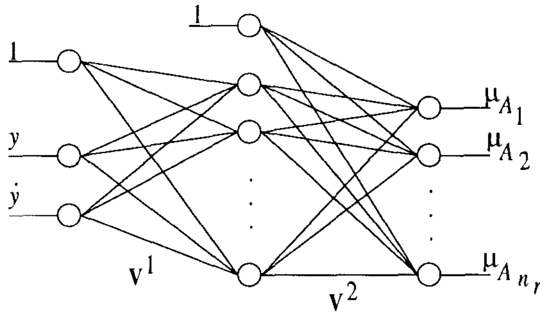

Figure C.1: Structure of NN_μ

The matrices $\mathbf{V}^1$ and $\mathbf{V}^2$ represent weight parameters of the neural network. The circles represent nodes, and the lines represent a connection between nodes and have weights associated with them. For example, the line between the i-th input node and the j-th hidden node has a weight denoted as v_{ij}^1 (the ij-th element of the matrix $\mathbf{V}^1$). An isolated node, thej-th hidden node for example, has a structure as seen in Fig. C.2.

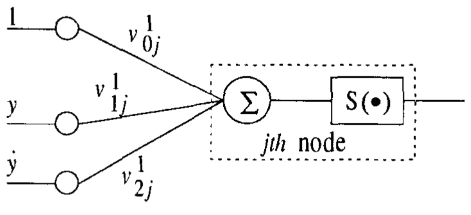

Figure **C.2:** The Input and Output Structure **of** thej-th Node **of** the Hidden Layer

The output of the j-th node is $S\left(v_{0j}^1 + y v_{1j}^1 + \dot{y} v_{2j}^1\right)$, where $S(\bullet)$ is called an activation function. The activation function is essentially a smooth threshold function, and we chose to define this function as

$$S(x) = \frac{1}{1+e^{-x}} \tag{C.1}$$

$S(x)$ approaches 1 as x approaches ∞, approaches 0 as x approaches $-\infty$, and $S(x) = .5$ when $x = 0$. Note that the input layer (as well as the hidden layer) has a node of input value 1 (Fig. C.1). Thus, the negative of the weight v_{0j}^1 represents the threshold of $y v_{1j}^1 + \dot{y} v_{2j}^1$. Referring to Fig. C.1, we can use the above definitions to propagate forward through the neural network. If we let the outputs of the hidden layer nodes be $\boldsymbol{h}$, and let n_h be the number of hidden nodes which receive information from the input layer, we have

$$h_j = \left\{ \begin{array}{cc} 1 & ;\, j = 0 \\ S\left(v_{0j}^1 + y v_{1j}^1 + \dot{y} v_{2j}^1\right) & ;\, j = 1,2,\ldots n_h \end{array} \right\} \tag{C.2}$$

Noting that n, is the number of rules and thus, the number of outputs NN_μ, we continue propagating through the neural network to get

$$\mu_{A_s} = S\left(\sum_{j=1}^{n_h} \quad \right) \quad \text{for } s = 1, 2, \ldots n, \tag{C.3}$$

Thus, Eqn. (C.2) and (C.3) smoothly map (i.e. the mapping is differentiable) the input $[y\ \dot{y}]^T$ to the output of *NN, ,* $\{\mu_{A_s}\}_{s=1}^{n_r}$, parameterized by the weights denoted by the elements of the matrices $\mathbf{V}^1$ and $\mathbf{V}^2$.

The next step is to adapt these weights such that the output of NN_μ approaches the output of a set of training data. The training data for NN, is obtained after clustering a set of n_t data points, $[y\ \dot{y}]_i^T$, of the input space into n_r regions, R_s for s = 1, ... n_r. We denote the n_r dimensional output vector of the training data for NN_μ as w_i^s, where

$$w_i^s = \begin{pmatrix} 1;\ x_i \in R_s \\ 0;\ x_i \notin R_s \end{pmatrix} \qquad \text{for} \quad i = 1, \ldots n_t; \quad s = 1, \ldots n_r \tag{C.4}$$

For example, if the input point $[y\ \dot{y}]_*^T$ belonged to R_2, the corresponding output vector for the training data would be $\mathbf{w}_* = [0\ 1\ 0\ \ldots 0]^T$.

In order to make the output vector of NN_μ, denoted as μ_{A^*}, approach the output vector of the training data, $\mathbf{w}_*$, we use the back propagation method to adjust the weights of NN_μ to minimize the Euclidean norm of the error between these two terms. Given a data point, *, we denote the quantity to be minimized as

$$J = \frac{1}{2}\sum_{s=1}^{n_r}\left(\mu_{A_s} - w_*^s\right)^2 = \frac{1}{2}\left(\mu_{A^*} - w_*\right)^T\left(\mu_{A^*} - w_*\right) \tag{C.5}$$

Our objective is to minimize J with respect to the weights of *NN, .* Noting that the vector, $\mathbf{w}$,, is not a function of the weights of NN_μ, we find the gradient of J with respect to each of the weights, for example, v_{ij}^1. The gradient is

$$\frac{\partial J}{\partial v_{ij}^{1}} = \left(\mu_{A^*} - w_*\right)^T \frac{\partial \mu_{A^*}}{\partial v_{ij}^{1}} \tag{C.6}$$

Since $S(\bullet)$ is smooth, this gradient exists and can be found by propagating backwards through *NN*,. The weight, v_{ij}^{1}, can be adjusted by

$$v_{ij}^{1}(t+1) = v_{ij}^{1}(t) - \alpha \frac{\partial J}{\partial v_{ij}^{1}} \tag{C.7}$$

where ***a*** is the training rate and t is the iteration step of training.

The neural networks, NN_c^s, have the same multilayered neural network structure and back propagation training method.

Nomenclature

y:	lateral error from magnetic sensor to the center of the road
y :	lateral velocity with respect to the center of the road
6,:	front wheel steering angle command
δ_c :	change in front wheel steering angle command
$\dot{\varepsilon}$:	yaw rate of vehicle
$\dot{\varepsilon}_d$:	desired yaw rate
t,:	current time
t,:	look-ahead preview time
t_2:	time that the vehicle will reach the next curvature transition
$\hat{y}_{t_l}(t_c)$:	estimate of y, at time $t,$ ahead of the current time, t_c
$Y_{d,t_l}(t_c)$:	desired lateral error at time t_l ahead of the current vehicle position
p:	radius of curvature of the roadway
n:	subscript denoting the *next* curve
c:	subscript denoting the *current* curve
v:	longitudinal velocity
LE:	linguistic variable for lateral error, *y*
CLE:	linguistic variable for change in lateral error, cy
YWR:	linguistic variable for yaw rate relative to desired yaw rate, (E − B)
CA:	linguistic variable for change in front wheel steering angle command, δ_c
n_r:	number of rules
n_t:	total number of training data (TRD)
x:	input vector to NN_μ and NN_c^s
X:	linguistic variable for **x**
δ_t^i:	the front wheel steering angle of the i-th training data (TRD) point, this is used for target data for NN_c^s
δ_c^s :	the front wheel steering angle command inferred by NN_c^s
w_i^s :	the i-th target data point for s-th output of NN_μ

References

[1] Araki, S., Nomura, H., Hayashi, I., "Self-generating Method of Fuzzy Inference Rules", *International Fuzzy Engineering Symposium* (IFES'92), 1992, pp. 1047.

[2] Dickerson, J.A., Kosko, B., "Ellipsoidal Learning and Fuzzy Throttle Control for Platoons of Smart Cars", Book Chapter of <u>An Introduction to Fuzzv Logic Applications in Intelligent Svstems</u>, edited by Yager, R., and Zadeh, L.A., *Klewer Academic Press,* 1992.

[3] Dickerson, J.A., Kosko. B., "Fuzzy Longitudinal Control for Car Platoons", *Modern Tools for Manufacturing Systems; Proc. of the International Workshop on Emerging Technologies for Factory Automation*, 1992.

[4] Godthelp, H., "Vehicle Control During Curve Driving", *Human Factors,* Vol. 28, No. 3, 1986.

[5] Hessburg, T., Tomizuka, M., "A Fuzzy Rule Based Controller for Lateral Vehicle Guidance", Publication of PATH project, ITS, UC Berkeley, UCB-ITS-PRR-91-18, August 1991.

[6] Hessburg, T., Peng, H., Tomizuka, M., Zhnng, W., "An Experimental Study on Lateral Control of a Vehicle", Publication of PATH project, ITS, UC Berkeley, UCB-ITS-PRR-91-17, August 1991.

[7] Hessburg, T., Tomizuka, M., "Experimental Results of Fuzzy Logic Control for Lateral Vehicle Guidance", to appear as a Publication of PATH project, ITS, UC Berkeley.

[8] Jang, R., "Self-Learning Fuzzy Controllers Based on Temporal Back Propagation", IEEE *Transactions on Neural Networks,* Vol. **3,** No. **5,** 1992.

[9] Lee, M., and Takagi, H., "Integrating Design Stages of Fuzzy Systems using Genetic Algorithms", *Proc. **of** IEEE 2nd Int'l Conference on Fuzzy Systems (FUZZ-IEEE'93)*, Vol. 1, 1993, p. 612.

[10] Nomura, H., Hayshi, I., and Wakami, N., "A Self-tuning Method of Fuzzy Control by Descent Method", *4th* IFSA *Congress,* Vol. Engineering, July, 1991, pp. 155.

[11] Peng. H., Tomizuka, M., "Vehicle Lateral Control for Highway Automation", 1990 *American Control Conference,* San Diego, CA, pp. 788.

[12] Peng, H., Tomizuka, M., "Optimal Preview Control for Vehicle Lateral Guidance", *Publication of PATH project, ITS,* UC Berkeley, UCB-ITS-PRR-91-16, August 1991.

[13] Rumelhart, D. E., Hinton, J. E., Williams, R. J., "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing* (Rumelhart, D. E. and McClelland, J. L. editors), MIT Press, Cambridge, MA, 1986, pp. 318.

[14] Takagi, H., Hayashi, I., "NN-driven Fuzzy Reasoning", *International Journal* of *Approximate Reasoning* (Special Issue of IIZUKA '88), Vol. *5,* NO. **3,** 1991, pp. 191.

[15] Zhang, W., Parsons, B., West, T., "Intelligent Roadway Reference System for Vehicle Lateral Guidance/Control", 1990 *American Control Conference,* San Diego, CA.