Neural Network based Controller for High Speed Vehicle following Predetermined Path

by

Tan Zhang Yaw

Dissertation submitted in partial fulfilment of
the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

December 2006

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Neural Network based Controller for High Speed Vehicle following Predetermined Path

by

Tan Zhang Yaw

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,

[illegible]

NOOR HAZRIN HANY BT. MOHAMAD HANIF

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SERI ISKANDAR
TRONOH, PERAK DARUL RIDZUAN

December 2006

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

TAN ZHANG YAW

ABSTRACT

The actual integration of automated control systems in vehicles such as Anti-lock Braking Systems (ABS) or Traction Control System (TCS) has proved to increase road safety and improve driver's comfort. Since most of the accidents are attributed to the fault of the driver, automated control systems in vehicle safety technology may dramatically better road safety by improving driver's performance. This thesis presents an enhanced and improved autonomous intelligent cruise control systems with obstacle collision avoidance integrated with path following/lane keeping. Obstacle collision avoidance is the ability to avoid obstacles that are in the vehicle's path, without causing damage to the obstacle or vehicle. Path following/lane keeping is the ability to follow the vehicle's path and keeping in its lane, as accurately as possible. The idea is to have a vehicle that drives by itself and avoids obstacles in the real world. Every instant, the vehicle decides by itself how to modify its direction according to its environment. This thesis demonstrates Gaussian functions and multi-objective cost function employed alongside with the Neural Network and optimal preview controller for control of the position of the vehicle to move while avoiding collision with obstacles. Each obstacle is represented independent of the others as a bell-shaped hump by the Gaussian functions which serve as an obstacle recognition system. Multi-objective cost function is formed for the planning strategy to generate, evaluate and select plans so that the vehicle can select which direction to move. Neural Network and optimal preview steering control are utilized to control a full linear steering model of a vehicle so as to increase path following accuracy. Optimal preview control is capable to portray the driver's vision of the path and process the knowledge while Neural Network controller has the ability to 'learn' from past errors and adjust the network to obtain specific target output. In this thesis, a MATLAB simulation environment was created to simulate the ability of a vehicle to avoid obstacles that are in the vehicle's path. Simulated obstacle avoidance has confirmed the capability of a vehicle to precisely avoid collision with obstacles while traveling on high speed along its predetermined path.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisor, Ms. Noor Hazrin Hany bt. Mohamad Hanif, for giving me this wonderful opportunity to work on such an interesting project. Allow me also to express my deep appreciation and profound thanks to my supervisor for her meritorious assistance, guidance and perseverance throughout this project. It is through the help of her that this project was possible thus leading to an ultimate success. I will always value the excellent scientific, engineering and ethical qualities that she has tried to instill in me, through leading by example. I am grateful to her for always being accessible and for providing a friendly and stimulating environment to work on this project.

I would like to express my heartfelt appreciation and ardent gratitude to Universiti Teknologi PETRONAS (UTP) Electrical and Electronics Engineering Department lab technologists, lab executives and lecturers for their approachable demeanor and for their unconditional help and effort in making my project a success. I would also like to thank my fellow colleague Sim for his relentless assistance and willingness to oblige when I needed help. Nevertheless, a special thanks to all my peers at UTP for their comments, suggestions and encouragement.

I want to thank my family for their support, prayers and encouragement. I especially want to extend my thanks to Siew Hui for standing by me relentlessly.

Above all, I would like to thank God for continuing to bless me with amazing family and friends and providing me with countless opportunities throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background of Study

Malaysia is considered relatively safe for driving compared to other developing countries. But if we observe past years' and recent road accidents statistics/reports, we can perceive that the propensity of road accidents in Malaysia is on the rise. Accidents are generally classified as single vehicle accidents and multiple vehicle accidents. Single vehicle accidents in which the vehicle is either colliding with fixed objects or with pedestrians or the vehicle may fall in a ditch and so forth whereas multiple vehicle accidents in which two or more than two vehicle can either collide head–on or one vehicle may collide with the front vehicle at the back or may a have side-swipe type collision. As a rapidly developing country, Malaysia has seen a dramatic increase in the number of vehicle ownership, averaging 8 % per annum from 7.7 million in 1996 to 12.8 million vehicles in 2003. This had translated into an increase in the number of road accidents from 189,109 cases in 1996 to 298,651 cases in 2003. At the same time, the number of fatalities resulting from road accidents had decreased slightly from 6,304 cases in 1996 to 6,282 cases in 2003. However, the number of deaths per 10,000 vehicles had decreased from 8.2 deaths in 1996 to 4.9 deaths per 10,000 in 2003 [1]. According to [2], the main cause of road accidents on the Malaysian roadways is the loss control of vehicle (*Terbabas sendiri (satu kenderaan)*) which contributed 23.19 % of the total road accidents in 2003 (see Table 1.1 in Appendix A). A comprehensive study of road safety by [3] found that human error was the sole cause in 57 % of all road accidents and was a contributing factor in over 90 % [4].

In the vehicle safety technology arena we have seen major advances in the last few decades. These developments from three-point seatbelts to airbags to Anti-lock Braking Systems (ABS) have saved thousands of drivers' and passengers' lives. And recently, it has gotten even better. Accelerated by the advent of the microchip, in the last ten years we have witnessed an incredible array of safety devices installed in vehicles. The actual integration of automated control systems in vehicles such as Traction Control System (TCS), Acceleration Slip Regulation (ASR) traction control, Electronic Stability Control (ESP), Autonomous Intelligent Cruise Control System has proved to increase road safety and improve driver's comfort. Driving is thus more relaxing, less stressful and safer and the driver can more easily concentrate on other important factors. Autonomous control system is a discipline in which control algorithms are developed by emulating certain characteristics of intelligent biological systems. It is quickly emerging as a technology that has opened avenues for significant advances in many areas. One of the many areas is vehicle safety technology. Looking at the genuine achievements of this technology, implementation is designed to reduce the number of accidents and fatalities on the roadways and highways. Automobile makers are currently trying to adopt this technology to the cruise control of their vehicles. However being autonomous, the degree of safety and reliability of this technology on the roadways is inevitably highly debated and questioned. This project employing Gaussian functions and multi-objective cost function alongside with optimal preview controller and Neural Network controller, as the heart of the vehicle autonomous control system would eventually open up possibilities of a whole new driving experience with vehicle traveling safely, accurately, precisely and providing a comfortable ride autonomously.

1.2 Problem Statement

With traffic continually increasing, basic cruise control is becoming less useful and becoming obsolete. This project has been conducted to enhance and improve the functions of conventional cruise control, a system which presently only maintains preset speed of a vehicle. The driver sets the speed and the system will take over the throttle of the vehicle to maintain the same speed. The existing cruise control system which only maintains the desired speed preset by the driver without any features assuming some of the driver's responsibility for safe driving is fraught with liability pitfalls and it is addressed as follows:

1.2.1 Safety

Most cruise control systems do not allow the use of the cruise control below a certain speed (normally 80 km/h) to discourage use in city driving. Therefore, when the cruise control is engaged, the vehicle is assumed to travel on a constant high speed. A vehicle traveling on high speed requires no margin of errors as any of it poses a high risk to the passengers and other users on the road. Although an error-free condition is not realistic, it should be as minimal as possible to ensure a level of safety for the utilization of an autonomous vehicle on the roadways.

1.2.2 Comfort/Convenience

Roads are often not obstacle and traffic-free. Traveling on a preset constant speed on a populated roadway poses a potential collision with the vehicle upfront or objects on its way. In dense traffic on the motorway and expressways with road environment constantly changing, conventional cruise control is often useless. Standard cruise control will not prevent accidents. It cannot provide steering assist when a driver is slow to take evasive action when unexpectedly confronted with another vehicle or object appearing in the vehicle's path. As a vehicle approaches a curve, this system is unable to provide compensatory steering assist to keep the vehicle in its lane and keep the vehicle well on its path.

1.3 Objective and Scope of Study

This project is an expansion of existing cruise control system. The system presented in this thesis not only maintains the desired speed preset by the driver, but also, whenever required provides steering assist on the motorways and expressways or country roads. This system relieves the driver of the permanent and monotonous chore of constantly adjusting the vehicle's speed and controlling the vehicle's steering. Also, it assumes some of the driver's responsibility for safe driving. It provides steering assist when a driver is slow to take evasive action when unexpectedly confronted with another vehicle or object appearing in the vehicle's path. At the start of evasive action, the system provides steering assist to help the driver avoid the obstacle. During evasive action, the system provides a safe and comfortable gap from the obstacle to help prevent the driver from getting too close to the obstacle. After evasive action, the system provides steering assist if the driver is slow to return the vehicle to its original course, helping prevent the vehicle from spinning out of control. In short, this system assists drivers in taking evasive action and then helps stabilize the vehicle. This project has been conducted with reference to the works of [5] on Neural Network based controller for high speed vehicle following predetermined paths. The aim of this project is to develop an improved and enhanced autonomous intelligent cruise control system that is capable to follow accurately the predetermined path with the smallest possible error and also to avoid obstacles along the path. For the ease and simplicity of the project, it will be assumed that the vehicle has perfect information about where the obstacles are (e.g., via radar sensor system, laser sensor system or computer vision system). The test speeds in interest are those commonly used on Malaysian motorways and expressways which are from 80 km/h to 110 km/h. Traveling above 110 km/h is well over the expressway legal speed limit and would not only earn the driver a speeding ticket but poses high risk to the passengers and other users on the road. In line with the objective to develop an autonomous intelligent cruise control system for high speed vehicle, therefore vehicle speed up to 200 km/h is studied to examine the reliability and robustness of the system. To make it easier to simulate path following/lane keeping with obstacle avoidance system of the vehicle, a simple and user friendly Graphical User Interface (GUI) is designed.

1.4 Outline of Thesis

This thesis is divided into several chapters and structured in the following way.

Chapter 2 (Literature Review And/Or Theory) discusses prior work related to collision avoidance, reviews Neural Network; its definition, training method (online) and algorithm (gradient descent) and outlines generic planning steps for obstacle avoidance.

Chapter 3 (Methodology/Project Work) recaps the works by Sharp [19] and Dandré [12] on the linear car model and road preview model. Gaussian functions and multi-objective cost function utilized for obstacle collision avoidance are described. This chapter also discusses optimal preview controller and Neural Network controller employed alongside with obstacle collision avoidance system to drive a linear car model accurately and precisely on prescribed paths.

Chapter 4 (Results and Discussion) discusses the simulation results of the vehicle traveling on high speed avoiding collision with obstacles while following the path designed to avoid the obstacles. The Graphical User Interface (GUI) developed for this purpose is also discussed. Path following/lane keeping with obstacle collision avoidance simulation results in the GUI for obstacle course (straight path, lane change and sudden change of direction) as well as path following simulation results in GUI for four predetermined obstacle free paths: sinus path, lane change, sudden change of direction and smooth random path are presented.

Chapter 5 (Conclusion and Recommendation) gives a conclusion of the work and proposes a range of possible future work that could improve the results of the thesis.

CHAPTER 2
LITERATURE REVIEW AND/OR THEORY

2.1 Relevant Research Topics

There has been much interesting research done for obstacle collision avoidance. Below are descriptions of a few relevant research topics.

In [6], there is a formalization of human centered design principles and illustrate their application using an automation system that assists drivers to avoid unsafe lane departures. This paper recognizes the importance of the human-computer interaction as related to collision avoidance. The safety and effectiveness of a collision avoidance system in an automobile is not only related to how well the automated system works, but how the entire human-computer system performs.

Making sensor-friendly vehicle and roadway systems would improve on the abilities of collision avoidance systems. In [7], work was done to show the improvements possible with complementary signal sensor and reflector technologies. These technologies can assist or replace single vehicle-based systems. The four most promising technologies passive license plates with enhanced radar return, roadside obstacle mounted radar-reflecting corner cubes, fluorescent paint for lane and obstacle marking, and light emitting diode brake light messaging are discussed especially on their improvement to the signal to noise ratio for the collision avoidance sensors. These sensor friendly systems should significantly improve collision avoidance systems.

In [8], a fuzzy logic enhanced car navigation and collision avoidance system has been designed. Essentially, the control of a car in this system is based on the flexible use of a fuzzy trajectory mapping unit that enables smooth trajectory management independent of car's initial position or position of the destination. This was done with a fuzzy controller consisting of 28 rules and a state machine containing 4 states.

For performing more demanding tasks, however, additional blocks of "intelligence" are required. The latter is quite possible thanks to the modular structure of the control system responsible for different task in separate without jeopardizing overall performance.

In [9], a multi-sensor collision avoidance system (CAS) is described in this paper. Measurements from radar, vision and sonar are combined using a fusion scheme that utilizes fuzzy clustering and estimation techniques to estimate relative motion between the vehicles. Fuzzy logic is used to generate audiovisual warnings for the driver. It also implements a throttle relaxer and brake actuator to slow the vehicle down. A prototype was implemented on a Humvee.

A fuzzy collision avoidance system for a fixed obstacle was designed and tested in [10]. This work describes a fuzzy trajectory controller with over 300 rules that is used with a specially designed car-driving robot. The rules were created based on the trajectories various drivers used to avoid a fixed obstacle. A laser was used as the obstacle detection device. While the robot and fuzzy controller worked successfully about 60 % of the time, the reasons for failure are understood.

Using Game Theory as a basis for collision avoidance is a subject of much research. One example would be from [11]. This work describes mathematically how an evader (car) can avoid a pursuer (moving obstacle or static obstacle), using non-cooperative game theory. There is no path to follow or limitation as to where the vehicle can go to avoid the obstacle, outside its own physical path restrictions.

2.2 Neural Network

This part of the chapter reviews briefly the implementation of Neural Network in the specific case of the control design. The learning process of a neural controller aiming to reduce a predefined cost function is introduced.

2.2.1 Overview of Neural Network

Neural Network is a powerful mathematical model originally designed to mimic a human's information processing structure and able to capture and represent complex input/output relationships. A Neural Network is a parallel processing structure composed of processing elements called neurons or nodes.

A neuron having multiple-input and single-output is illustrated in Figure 2.1 in Appendix A. Each neuron in an artificial Neural Network is based on:

- A set of input values a_i and associated weights w_i.
- A threshold or bias b.
- An activation function f, possibly non-linear, that operates on the weighted inputs and the bias and maps the results to an output u:

$$u = f\left(\sum_i w_i . a_i + b\right) \tag{2.1}$$

The Figure 2.2 in Appendix A depicts some common activation functions also called transfer functions. The network function of a neuron is determined largely by the connections between the neurons and depends on the location of the considered neuron in the network.

A Neural Network consists of a combination of neurons wired together in a complex communication network in one or several layers. For instance, a multi-layer feed-forward network is composed of input, hidden and output layers as depicted in Figure 2.3 in Appendix A. Each neuron is connected to the others neurons in the next layer through the weighting parameters.

The Neural Network's knowledge is stored within the inter-neuron connection strengths also called synaptic weights. Commonly the networks are adjusted, or trained, so that a particular input x leads to a specific target output $\hat{y}$.

This learning process proceeds by way of presenting the network with a training set $\{x(i), \hat{y}(i)\}$ composed of inputs together with the required response. A certain input is fed into the input layer of the network. The network will then produce an output. By comparing this output with the required target output, the error the network is making can be measured. This error can then be used to alter the connection strengths between layers in order that the network's response to the same input will be better the next time around. This can be done thanks to a cost function of the form:

$$J = \frac{1}{N}.\sum_{i=1}^{N} e(i) \tag{2.2}$$

$$e(i) = \frac{1}{2}\left(\hat{y}(i) - y(i)\right)^t.\left(\hat{y}(i) - y(i)\right) \tag{2.3}$$

and an optimization process aimed to reduce the cost function. The gradient method is commonly used but others methods exist. The update rule of the gradient method is:

$$w(new) = w(old) + \Delta w \quad (2.4) \qquad \text{with} \qquad \Delta w = -\gamma.\frac{\partial J}{\partial w} \quad (2.5)$$

where γ is the learning rate

and where w is a vector containing all weighting parameters

The true power of Neural Network lies in their ability to adapt to various situations. The learning process depends mainly on the modeling. This advantage allows to consider a wide range of outside conditions with little tuning and without the modification of the main structure of the process.

Neural Network is also powerful in the case of both linear and non-linear relationships when traditional linear methods become inappropriate as the plant to be controlled contains non-linearities [12].

Neural Network performs two major functions: learning and recall. Learning is the process of adapting the connection 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.

Learning rules of neural computation indicates how connection weights are adjusted in response to a learning example. The most used learning rule in engineering application is supervised learning. In this method, the Neural Network is trained to give the desired response to a specific input stimulus. The difference between actual output and desired response is known as error, which is used to adjust the connection weights.

Other learning rules are graded learning (output is 'graded' as good or bad on a numeric scale, and the connection weights are adjusted in accordance to the grade) and unsupervised learning (the network organizes itself internally so that each hidden neuron responds strongly to a different set of input stimuli) [13].

2.2.2 Neural Network Training

Neural Network 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. Online (incremental) training was used in this project and is outlined in this section.

2.2.2.1 Online Training

Online training updates weights and biases as each input is presented to the network. By setting any 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). Summarized in [14], the online training proceeds as follows:

Step 1: Initialize the weights.

Step 2: Process one training case.

Step 3: Update the weights.

Step 4: Repeat Step 2 onwards until the stopping criterion has been reached.

2.2.3 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 [15]. 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 [16]. Gradient descent method was used for this project and is described in this section.

2.2.3.1 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:

$$\Delta w = -\gamma \frac{\partial J}{\partial w} \tag{2.6}$$

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 [17]. 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 [15]. The learning rate is change in such a way that it will be increased if stable learning is obtained per instance or decreased when the learning becomes unstable [13].

2.3 Generic Planning Steps for Obstacle Avoidance

Essentially, planning is one approach that allows for more than simple reactions to what it sensed. It utilizes information about the problem and environment, often in the form of some type of model and considers many options and chooses the best one to achieve the closed-loop control objectives. Planning provides for a very general and broadly applicable methodology and it has been exploited extensively in conventional control (e.g., in receding horizon control and model predictive control). As compared to the fuzzy and expert system approaches, it exploits the use of an explicit model to help it decide what actions to take. Like the fuzzy and expert system approaches, it still, however, possible to incorporate heuristics that help to specify what control actions are the best to use. Hence, in a broad sense, planning approaches attempt to use both heuristic knowledge and model-based knowledge to make control decisions; this may be the fundamental reason for selecting a planning strategy over a simple rule-based one. It is often bad engineering practice to only favor the use of heuristics and ignore the information provided by a good mathematical model; planning strategies provide a way to incorporate this information.

"Action plans" are often formed to try to achieve specific goals. For instance, an "action hierarchy" given in Figure 2.4 is performed by the vehicle as one type of action plan. The goal is ultimately develop a simple planning strategy for control of the autonomous vehicle to move along its predetermined path while avoiding obstacles in its way.

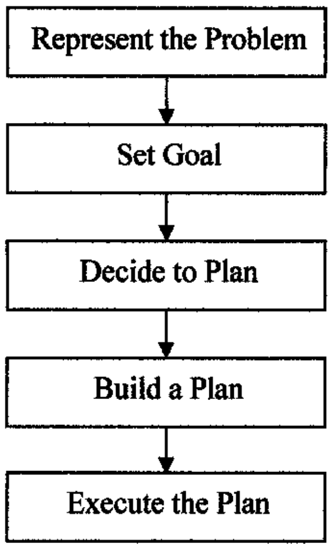


Figure 2.4: Generic planning steps

2.3.1 Represent the Problem ("Planning Domain")

In order to plan, some types of representation (model) of the problem that must be solved must exist. This model in this case is in the form of a road map where the vehicle is trying to plan a route to avoid obstacles along its predetermined path. Generally these models are thought as being acquired via experience (i.e., via learning), however it is certainly the case that instincts (model passed to humans via evolution) affect planning. For instance, humans have certain "hard-wired" knowledge that can be thought of as aspects of models that influence planning (e.g., tendency to have a fear of snakes and some insects). Performance in planning is critically dependent on the model of the problem. A poor model will generally lead to a bad plan, or at least to one that soon fails that is colliding with an obstacle or going off the path and thus requiring replanning. A high quality model that allows to project far into the future (or down a hierarchy of tasks and sub goals), may lead to better plans. However characteristics of the problem domain may make it impossible to specify a good model. For instance, time varying and stochastic features of some problem domains may make it impossible to predict into the future with much accuracy and hence make it a waste of time to predict too far into the future.

2.3.2 Set Goal

Setting goals is essential to planning, since without goals there is no purposeful behavior. Goals can be very different for different system, environment and times. Goals are driven by evolutionary characteristics (e.g., the goal of survival, the goal of reproduction), but in humans such goals can also be significantly affected by humans' values and ideals (e.g., ones set by culture). Goals can be learned and can consist of a time-varying hierarchy or sequence of sub goals. The goal of the vehicle is pretty much straight forward where it is required to avoid collision with obstacles along its predetermined path.

2.3.3 Decide to Plan

Sometimes humans simply react to situations without considering the consequences of their actions. Other people decide to develop a plan since they may think that this will allow them to more successfully reach their goals. There are many issues that affect the decisions of whether or not to plan (e.g., physiological and cultural). Many lower animals (e.g., some bacteria) cannot plan; they simply react to stimuli.

2.3.4 Build a Plan (Select a Strategy)

Normally the selection of a plan first involves projecting into the future using a model (e.g., in path planning of the vehicle in this case) and often involves considering a variety of sequences of tasks and sub goals to be executed. In terms of graph-theoretic view, this may be thought of as a "tree" of plans where the nodes of the tree are tasks or sub goals and links between these indicate plans (a path in the tree is a candidate plan). See Figure 2.5 in Appendix A. How "deep" a tree generate (e.g., how far to plan into the future) depends on the quality of the model, characteristics of the environment and how much time or resources that have to be planned. The second key component of selecting a plan is the solution of an optimization problem. For instance, suppose that the links on the "tree" that represents the set of possible plans are each labeled with integer values that represent the "cost" of performing the task represented by going in that direction in the tree. For instance, the cost may represent distance traveled or time executed the task and the characteristics of the cost are typically dictated by the goal. Next, supposedly the tree represents a finite number of possible plans and that the cost of a plan is represented by summing the costs of each link that represents a step in the plan. Then, the plans can be ordered according to the cost and minimization can be perforemed by picking the lowest cost plan (the "best" plan). Again, see Figure 2.5 in Appendix A. For example, this may be the shortest route to take for the vehicle to avoid all the obstacles, if it is solving the subtask of obstacle avoidance of the vehicle.

2.3.5 Execute the Plan, Monitor and Repair/Replan

After selecting a plan, how to execute the plan must be decided. While the plan is executed, it is monitored by detecting the deviations for what is expected to make sure that all is going well. Then, especially in an uncertain problem domain, it could be that there is a "plan failure" so that there is a need to repair the current plan, or to develop a completely new plan (the frequency of replanning is generally proportional to the amount of disturbances in the system). The decisions of whether to simply "tweak" the current plan, or develop a completely new one is difficult and can involve assessments of available resources (e.g., time) and the extent to which goals is being met. Some problem domains are particularly difficult to monitor and hence there may need to be a parallel process operating that estimates the "state" of the domain from available sensed information (this is sometimes called "situation assessment"). The ability to do this depends on the "observability" properties of the problem domain (i.e., whether the state of the plant can be computed from measured inputs and outputs). When using such estimates, the need to guess whether the plan is succeeding and subsequently replan arises [18].

CHAPTER 3
METHODOLOGY/PROJECT WORK

This chapter examines clear procedures of this project in developing the control of a high speed vehicle following a predetermined path or keeping in a designated lane and whenever required, avoiding collisions with obstacles in its path. The first part of the chapter describes the obstacle collision avoidance integrated with path following/lane keeping cruise control of a vehicle. The second and third part recaps the works by Sharp [19] and Dandré [12] on the linear car model and road preview model. The fourth part is devoted to developing the evasive control of the obstacle collision avoidance system and in the final part; the steering control of a vehicle aiming to follow prescribed paths/lane is examined.

3.1 Path Following/Lane Keeping with Obstacle Collision Avoidance

The system as in Figure 3.1 provides steering assist when a driver is slow to take evasive action when unexpectedly confronted with another vehicle or object appearing in the vehicle's path while traveling on a prescribed path/lane. As depicted in Figure 3.2 in Appendix A, if another vehicle or object is detected in the driver's path/lane (assumingly via radar sensor system, laser sensor system or computer vision system) while following a path or keeping in its lane, the system takes evasive action; steering sharply away from the other vehicle. This system assists with steering to support the driver. At the start of evasive action, the system provides steering assist to help the driver avoid the obstacle. During evasive action, the system provides a safe and comfortable gap from the obstacle to help prevent the driver from getting too close to the obstacle. After evasive action, the system provides steering assist if the driver is slow to return the vehicle to its original course, helping prevent the vehicle from spinning out of control. This system assists drivers in taking evasive action and helps stabilize the vehicle and then continues following its predetermined path or keeping in its designated lane.

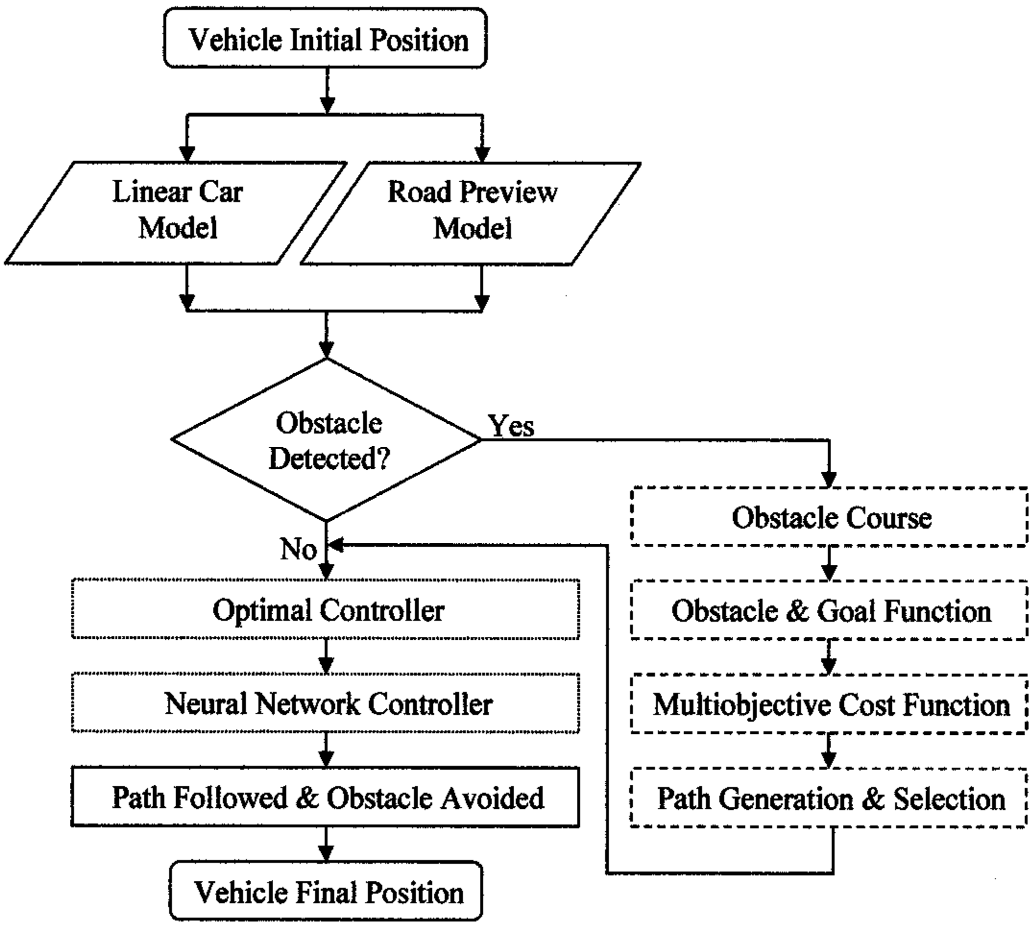


Figure 3.1: Path following/lane keeping system (dotted) integrated with obstacle collision avoidance system (dashed)

3.2 Linear Car Model

A standard yaw/sideslip model of a vehicle is shown in Figure 3.3 in Appendix A and has been described by Sharp and Valtetsiotis in [19] and in [20]. The model consists of a rigid body based on the following assumptions:

- Suspension is omitted and the car is a single rigid body.
- The car is moving on a level plane (the road is flat).
- The vehicle has three degrees of freedom: forward motion, lateral motion and yawing motion.

- 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.
- A constant forward speed u is considered and the input will be the steering wheel *angle* d_{sw}.
- The relation between the steering wheel movement and that of the front road wheels is fixed and defined by a gear ratio *G*. Inertial effects of steering the wheels are discounted.
- Aerodynamic forces are discarded.
- Tyre aligning moments are ignored.
- Lateral weight shift and roll are discounted.
- The car is neutral steering.

The model is very simple and cannot represent truly a car. The model includes the forward speed considered as constant, which strongly limits the real driving of a car and neglects the suspension and the load transfer, which are vital when negotiating a turn at high speed. In practical, 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. Considering the control of braking and thrust would imply the implementation of a second Neural Network. Still this model suits fairly well the specifications required, since the main purpose of the study is path following control. The vehicle model parameters to define the car can be found in Table 3.1 and Table 3.2 in Appendix A.

One can notice that since $a.C_f=b.C_r$ the car is neutral steering and that the mass of the body is quite heavy and the yaw inertia is also significant. The centre of gravity is towards the front of the car.

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

$$x = [x_1 \quad x_2 \quad x_3 \quad x_4]^T \tag{3.1}$$

where x_1 is global lateral position y

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

x_3 is global lateral angle Ψ

x_4 is global attitude rate $\dot{\Psi}$

and

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -(C_f + C_r)/Mu & (C_f + C_r)/M & (bC_r - aC_f)/Mu \\ 0 & 0 & 0 & 1 \\ 0 & (bC_r - aC_f)/I_z u & (aC_f - bC_r)/I_z & -(a^2C_f + b^2C_r)/I_z u \end{bmatrix} \tag{3.2}$$

$$B = \begin{bmatrix} 0 \\ C_f / MG \\ 0 \\ aC_f / I_z G \end{bmatrix} \tag{3.3}$$

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 $\dot{x}(k+1) = A_d x(k) + B_d \delta_{sw}(k)$ 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. Detailed explanation on the linear car model could be retrieved from works in [12]. The preview points will be explained in the next section.

3.3 Road Preview Model

The purpose of the road preview is to represent very simply the road information stored and used by the driver through his/her eyes. In real environment and considering an unknown path, this information would come for instance from one or more sensors which would characterize the road path ahead of the vehicle. The sensor could for example estimate the relative locations of the white painted lines of a motorway. Then the controller would use the data gathered by the sensor(s) and the process would be on-line. In the case of a known path (racing circuit, etc.), the sensors are not needed anymore and the information can be directly stored in a computer. On-line computation is then not necessary.

Five paths are considered for the study: straight path, 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.

Taking n as the number of preview values, the lateral deviations at time kuT meters ahead of the car could be represented as $y_{ref}(k) = [y_{r0} \quad y_{r1} \quad .... \quad y_{rn}]^T$. The uT is the x spacing, in which u is the speed of the vehicle. Figure 3.4 in Appendix A 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.

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:

$$D = \begin{bmatrix} 0 & 1 & 0 & .... & 0 \\ 0 & 0 & 1 & .... & 0 \\ .... & .... & .... & .... & .... \\ 0 & 0 & 0 & .... & 1 \\ 0 & 0 & 0 & .... & 0 \end{bmatrix} \quad (3.4) \qquad \text{and} \qquad E = \begin{bmatrix} 0 \\ 0 \\ .... \\ 0 \\ 1 \end{bmatrix} \quad (3.5)$$

Detailed explanation on the road preview model could be retrieved from works in [12].

3.4 Obstacle Collision Avoidance System

This part of the chapter delineates the steering control of a linear vehicle aiming to follow a prescribed path or to keep in a designated lane as shown in Figure 3.5. This system supposedly uses on-board cameras and radar to detect when a vehicle suddenly appears from the side, for example at an intersection or unexpectedly confronted with another vehicle or object appearing in the vehicle's path while traveling on a prescribed path/lane as depicted in Figure 3.6 in Appendix A though in this project obstacle positions are assumed to be known. If the system determines that a collision may occur, it provides an evasive action; steering sharply away from the other vehicle or object. This system assists with steering to support the driver. At the start of evasive action, the system provides steering assist to help the driver avoid the obstacle. During evasive action, the system provides a safe and comfortable gap from the obstacle to help prevent the driver from getting too close to the obstacle. After evasive action, the system provides steering assist if the driver is slow to return the vehicle to its original course, helping prevent the vehicle from spinning out of control. This system assists drivers in taking evasive action, and then helps stabilize the vehicle. The first section is devoted to developing the obstacle course and the characteristics of the vehicle. In the second section, the obstacle and goal functions are described. The third and fourth section respectively considers the multiobjective cost function and discusses how the planning strategy generates, evaluates and selects plans so that the vehicle can select which direction to move.

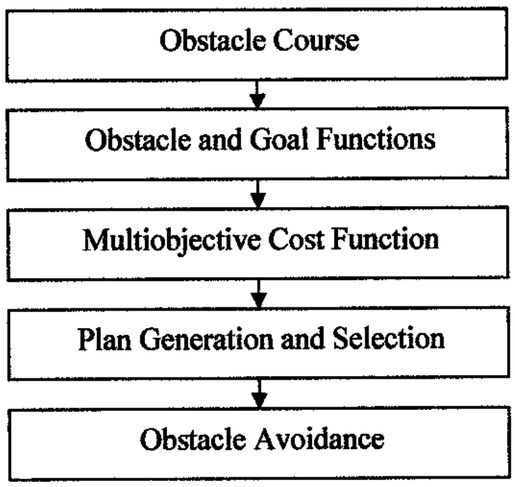

Obstacle Course
Obstacle and Goal Functions
Multiobjective Cost Function
Plan Generation and Selection
Obstacle Avoidance

Figure 3.5: Obstacle collision avoidance system

3.4.1 Obstacle Course and Vehicle Characteristics

It is assumed that perfect information about where obstacles are, is available with known (x, y) positions. A test field of x-coordinate, $x \in [0, 30]$ and the y-coordinate, $y \in [0, 50]$ with poles like obstacles are shown from a top view in Figure 3.7. The intended path of the vehicle is a straight line with the initial vehicle position at (1, 15) and that the goal function position at (49, 15) as shown in Figure 3.7 via "□" and "x" respectively.

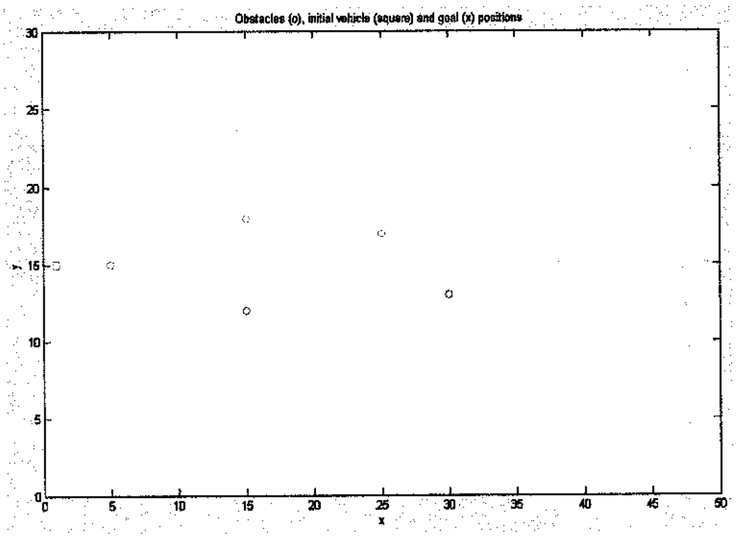


Figure 3.7: Initial vehicle position, goal position and obstacles

Six poles shown in Figure 3.7 that are at positions (5, 15), (15, 18), (15, 12), (25, 17), (30, 13) and (38, 15) respectively. The vehicle knows its own position (assumingly via radar sensor system, laser sensor system or computer vision system) and the goal position that it seeks to move to.

When a vehicle decides to move from one position to another position, it can approximately do so in one time step. In particular the vehicle's current positions is ($x(k)$, $y(k)$) and the onboard computer commands it to move at an angle θ a distance of λ (see Figure 3.8 in Appendix A), it does so according to:

$$\begin{bmatrix} x(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \end{bmatrix} + \lambda \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \Delta\lambda \begin{bmatrix} \cos(\Delta\theta) \\ \sin(\Delta\theta) \end{bmatrix} \tag{3.6}$$

where the sum of the first two terms on the right side of the equation represent the desired position. λ is chosen to be 0.1. The last term is a noise term that represents effects on uncertainty that result in the vehicle not perfectly achieving the desired position. $\Delta\lambda$ is chosen to be a random number at each time step uniformly on [0.1λ, 0.1λ] representing that there is a 10 % uncertainty in achieving the commanded radial movement. Also, $\Delta\theta$ is assumed to be uniformly distributed on [$-\pi$, π]. Hence, when the vehicle is commanded to go to a particular position in one step time, it ends up somewhere in a circular region of radius 0.1λ around the desired position. Notice that in order to make such movements, the vehicle needs to sample its own position at each time step. Hence, feedback control is used in the following way of guidance: the current position is sensed and the command is made to move the vehicle to the new position. The vehicle may not end up where it was commanded to go, but at the next time instant, the vehicle will sense its position and make adjustments from that point and so on.

3.4.2 Obstacle and Goal Function

It should be clear that since it is assumed that the positions of the vehicle and all the obstacles are known, there is no need for a sensor that measures proximity to, or characteristics of obstacles. In a certain sense, a perfect model of a part of the environment is available. A perfect model of the entire environment is unavailable due to the uncertainty in reaching a desired commanded position. The information in Figure 3.7 is represented and utilized about where the vehicle starts, where it should go and where the obstacles are. Since a planning strategy is used, it is critical to realize that the path-finding problem is needed to be formulated as an optimization problem. To do this, the simple approach of constructing a surface (sometimes called a "potential field") that represents where the obstacles are is taken. In particular, to represent the obstacles in Figure 3.7, the Gaussian functions of unity height is taken and centered at each of the obstacles and an "obstacle function" $J_o(x, y)$ that is the maximum value of each of those functions at each point (x, y) as shown in Figure 3.9 (the use of the maximum of the six Gaussian functions representing the six obstacles, rather than, for instance, simple addition of the six Gaussian functions, ensures that each obstacle position is represented independent of the others) is computed.

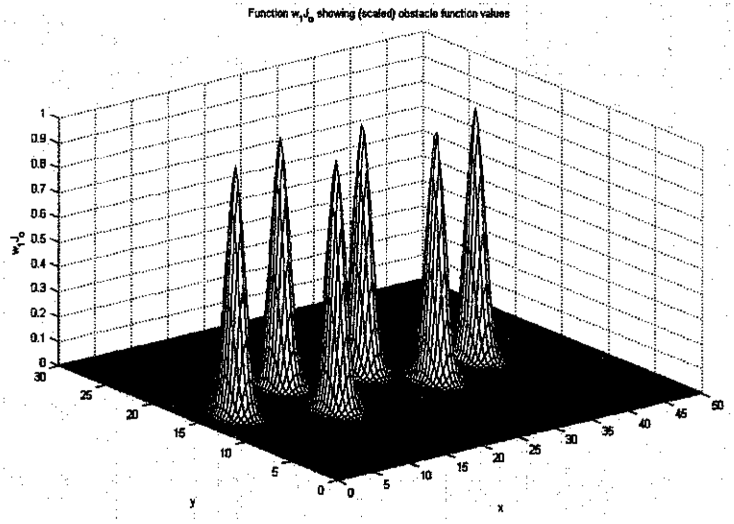


Figure 3.9: Obstacle function $J_o(x, y)$ (scaled by ω_l)

In Figure 3.10 the contour plot $J_o(x, y)$ along with the initial vehicle position and goal position is shown.

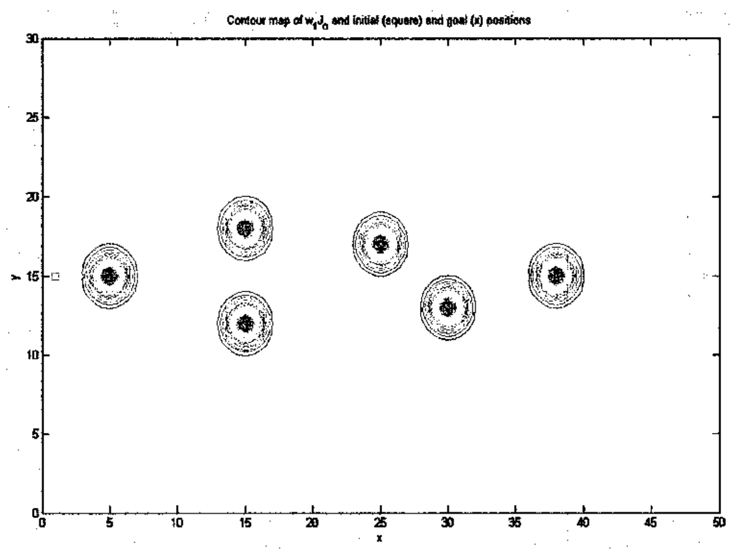


Figure 3.10: Obstacle function $J_o(x, y)$ (scaled), contour form, with initial vehicle position and goal position

The contour nicely shows the "spreads" (variances) of the Gaussian functions and that there is a type of overlap such that the values of $J_o(x, y)$ are at least a bit above zero for any positions where the vehicle should not be in order to avoid collision with obstacles. Also, the obstacle function is scaled with a positive constant $\omega_1 > 0$ in the planning strategy, however here $\omega_1 = 1$ is chosen. Note that if the vehicle is moved about the environment in a way that the vehicle position is moved to points that try to minimize $J_o(x, y)$ (e.g., via hill climbing), then the vehicle will avoid the obstacles, due to the tails of the Gaussian functions. For many vehicle initial positions, the vehicle would move to the edge of the region and when it arrives there, it is always kept on the edge.

Next the goal being at the position (49, 15) is represented. To do this, the minimum point of a quadratic (bowl) function as follows is placed:

$$\omega_2 J_g(x,y) = \omega_2 \left[[x,y]^T - [49,15]^T\right]^T \left[[x,y]^T - [49,15]^T\right] \qquad (3.7)$$

where $\omega_2 > 0$ is a scale factor and $\omega_2 = 0.0001$ is chosen that will multiply this function. The scaled function is shown in Figure 3.11 as a contour plot. If at each time step the vehicle moved to go down the surface, it will move toward the goal, but it may run into an obstacle.

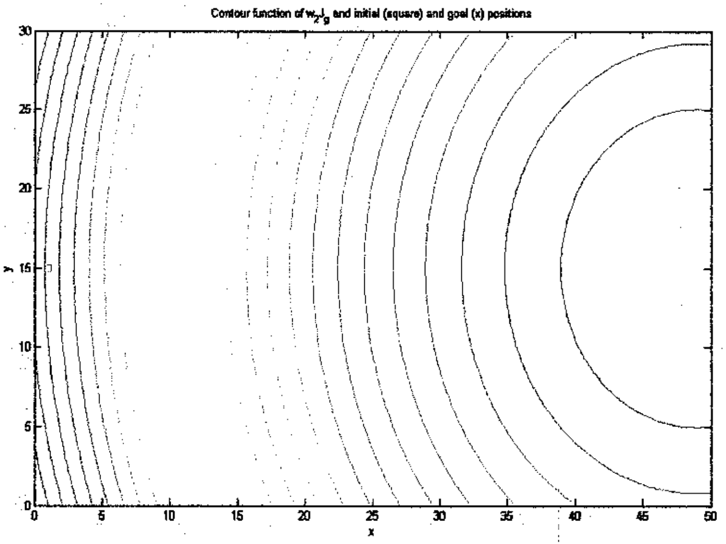


Figure 3.11: Goal function $\omega_2 J_g(x, y)$, contour form, with initial vehicle position and goal position

3.4.3 Multiobjective Cost Function

Firstly, a cost function is formed for the planning strategy to generate, evaluate and select plans so that the vehicle can select which direction to move. If the vehicle is commanded to move a distance of λ in a direction θ that is chosen by simply moving in the "direction of the steepest descent" on the function $J_o(x, y)$, then the vehicle would avoid obstacles but not reach the goal position and stay there. Similarly, if the direction was chosen to be the one with steepest descent for the $J_g(x, y)$ function, then it would move to the goal position but may collide with some obstacles for some initial vehicle positions.

To solve this problem, a "multiobjective cost function" is used (actually a special case where a "scalarization" approach is used to form a multiobjective cost, which is one of many ways to generate a Pareto cost)

$$J(x,y) = \omega_1 J_o(x,y) + \omega_2 J_g(x,y) \tag{3.8}$$

shown in Figure 3.12 where the weights ω_1 and ω_2 specify the relative importance of achieving obstacle avoidance and reaching the goal.

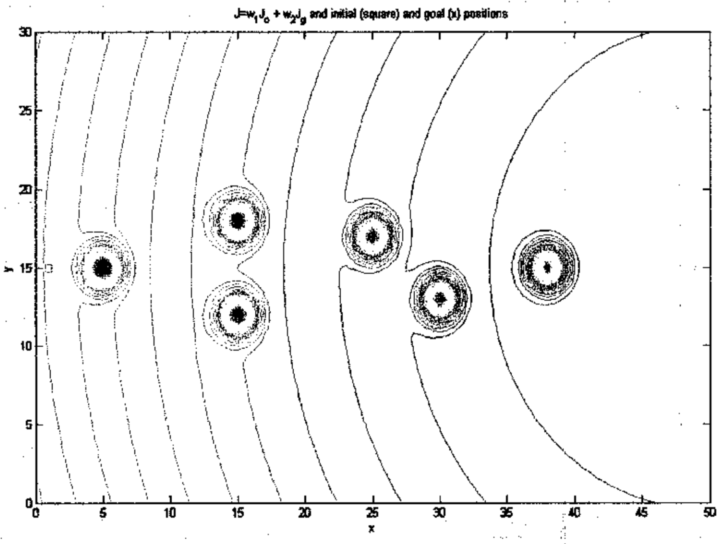


Figure 3.12: Multiobjective cost function $J(x, y)$ for evaluating plans

The magnitudes of the values of each term in selecting these must be taken into consideration. The choices of weight values above represent that obstacle avoidance is important, but the vehicle also must keep moving toward the goal position. The choice of the weights will affect the shape of the trajectory that the vehicle will move on toward the goal position.

3.4.4 Plan Generation and Selection

A simple approach is taken to plan generation and evaluation. If the vehicle is at a position (x, y), the value of J is computed at N_s values (x_i, y_i), $i = 1, 2, \ldots, N_s$, regularly spaced on a circle of radius r around the vehicle position (see Figure 3.8 in Appendix A, where $N_s = 8$). Here, $r = 1$ and $N_s = 16$ are used. This generates 16 plans, where one step is predicted ahead. More values of J can be computed that are along other longer paths. The set of plans is viewed as "the vehicle is at (x, y), move it to (x_i, y_i)." The plan is chosen to execute by finding a value i^* such that:

$$J(x_{i^*}, y_{i^*}) \leq J(x_i, y_i), i = 1,2,\ldots,N_s \tag{3.9}$$

(i.e., by finding the direction which will result in minimization of the multiobjective cost function). This direction $\theta(k)$ is called and the vehicle is commanded to take a step of length λ in the direction $\theta(k)$.

The above approach will approximate the "steepest descent approach" (hill-climbing) discussed above but analytical gradient information is unnecessary since the gradient of the multiobjective cost function is not explicitly computed. Higher values of N_s cost more computations in plan generation and evaluation, but they also provide more precise directional commands. Notice that by using the above strategy, for any initial position on Figure 3.12, it is expected that the vehicle will navigate so as to avoid the obstacles and move toward the goal by simply moving down the surface [18].

3.5 Path Following/Lane Keeping System

This part of the chapter is an account of [19] which examines the steering control of a linear vehicle aiming to follow a prescribed path or to keep in a designated lane as shown in Figure 3.13 in Appendix A. Ideally, as the vehicle approaches a curve, this system will use information from the vehicle's navigation or radar system to assess the curvature of the road and calculates the vehicle's appropriate speed. If the vehicle is traveling above that speed, the system applies the brakes to slow the car to the appropriate speed. However this feature is yet to be developed and would be recommended for future works. The first section considers the set up of an optimal preview controller and the final section outlines the Neural Network controller. Detailed explanation on the optimal controller could be retrieved from works in [12]. Figure 3.14 displays the system of path following/lane keeping of the vehicle.

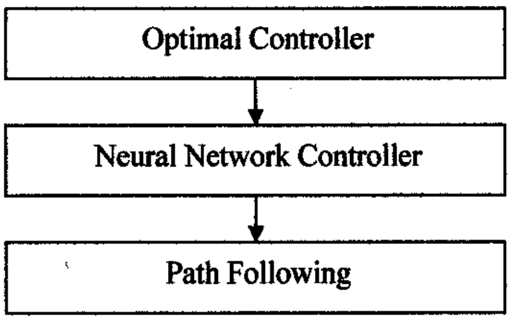


Figure 3.14: Path following/lane keeping system

3.5.1 Optimal Controller

The purpose of the controller implementation is to establish a connection between the road preview model and the car. Plainly, optimal controller is purported to represent and synthesize well the vision of the driver so that the vehicle can follow the path as accurately as possible. The structure combining the car and road preview models with the optimal controller is illustrated in Figure 3.15 in Appendix A. 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:

$$\begin{bmatrix} x(k+1) \\ y_r(k+1) \end{bmatrix} = \begin{bmatrix} A_d & 0 \\ 0 & D \end{bmatrix} . \begin{bmatrix} x(k) \\ y_r(k) \end{bmatrix} + \begin{bmatrix} 0 \\ E \end{bmatrix} . y_{ri} + \begin{bmatrix} B_d \\ 0 \end{bmatrix} . \delta_{sw} \tag{3.10}$$

According to simulation results obtained by [19] and repeated by [12], as the speed of vehicle is increased, the preview gain will be more oscillatory. Figure 3.16 in Appendix A shows the simulation result for the optimal preview gains of path following [13]. Detailed explanation on the optimal controller could be retrieved from works in [12].

3.5.2 Neural Network Controller

The purpose of the Neural Network controller is to mimic a driver's information processing structure and to capture and represent complex input/output relationship.

It is a learning and training strategy that trains and adjusts to converge to the desired output with the respective input given by comparing the output with the required target output. In short, it learns from past errors and adjusts the network to improve network's response.

Five paths (straight path, sinus path, 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 [19], it is proven that the optimal controller has the capability to precisely track reasonable paths.

Dandré [12] has continued the research by tracking the similar paths using Neural Network. The coefficients obtained through the optimal control theory were taken as the initial weighting parameters for the neural controller [13].

3.5.2.1 Implementation using Gradient Method

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 Dandré [12] in the following formula:

$$\delta_{sw} = w_1(k)z_1(k) + w_2(k)z_2(k) + \ldots w_{n+5}(k)z_{n+5}(k) \qquad (3.11)$$

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 are to 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. As the car is supposed to follow the simulated paths, the cost priorities are set as:

$$q_1=100,\ q_2=1,\ R_2=1 \text{ and } R_1=\begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix} \tag{3.12}$$

As it 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 taken 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 the [14], 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.

If the training mode of the network is set to be online, the weights will be updated each time the learning rate is updated [13]. Detailed explanation on the Neural Network controller could be retrieved from works in [12].

CHAPTER 4
RESULTS AND DISCUSSION

The simulation environment was created using MATLAB. Simulated obstacle collision avoidance has confirmed the capability of a vehicle to precisely avoid collision with obstacles along its prescribed path. It is then possible to integrate the path following/lane keeping system with the obstacle collision avoidance system for the vehicle to avoid obstacles whilst following its predetermined path. In the first part of the chapter, it is noticed that the results are promising where the vehicle is able to avoid obstacles along an obstacle course, without causing damage to the obstacle or the vehicle. The second part demonstrates the ability of the vehicle to accurately and precisely follow the path designed to avoid the obstacles on the obstacle course while traveling on high speed. The second part in addition considers the removal of the assumptions used in the previous chapter and the flaws of this system will be highlighted. The third part is dedicated to describe the functionality of the Graphical User Interface (GUI) developed which also presents the simulation results of path following/lane keeping with obstacle collision avoidance in GUI for obstacle course (straight path, lane change and sudden change of direction) as well as simulation results of path following in GUI for four predetermined obstacle free paths: sinus path, lane change, sudden change of direction and smooth random path.

4.1 Obstacle Collision Avoidance

Using the planning strategy as mentioned in the preceding chapter, obstacle course and vehicle, the trajectory shown in Figure 4.1 is obtained. Clearly, the vehicle moves so as to avoid the obstacles (via the effect of J) but tries to stay on course to the goal (via the effect of J_g). The effects of the uncertainty in reaching commanded positions is seen by the small deviations on the trajectory that are "corrected" at each step since the vehicle is assumed to get a measurement of its own position at each time step. Other vehicle paths result from other choices of obstacles and goal functions and their scale factors (higher weight on the goal function tends to reduce deviations away from obstacles). Moreover, a different pattern of points where the multiobjective cost function is evaluated can result in a different path. For instance, using fewer points on the circular patter results on trajectories that are not as smooth.

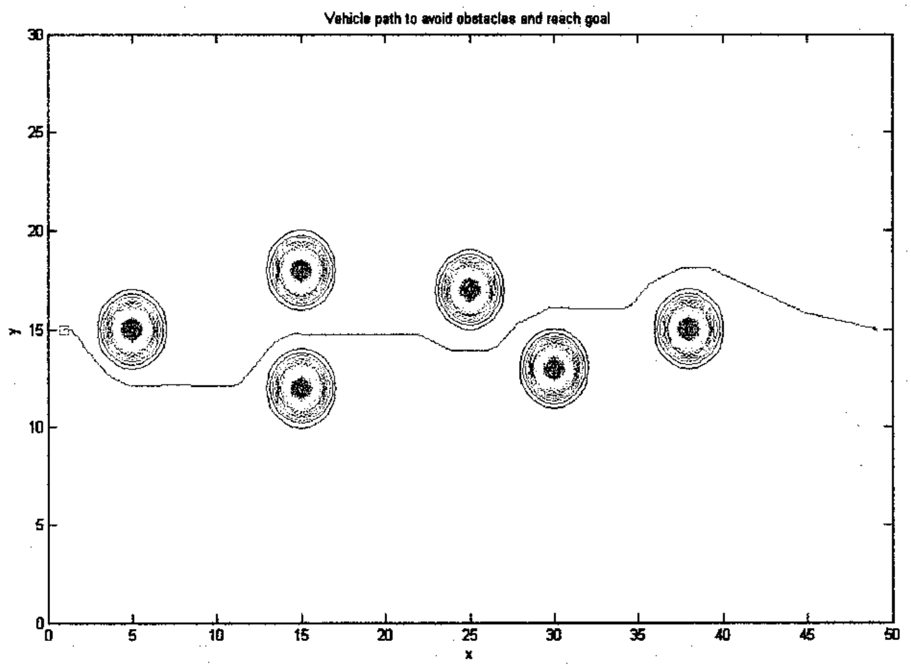


Figure 4.1: Vehicle path for obstacle avoidance and goal seeking

4.2 Path Following/Lane Keeping with Obstacle Collision Avoidance

With both system viz. path following/lane keeping and obstacle collision avoidance fused together as to form a whole, the reliability and robustness of the joint activity of the two systems is probed by simulating the path following on the obstacle course (straight path) with increasing vehicle speed as well as preview points. The test speed studied in this thesis is from 80 km/h to 200 km/h which is the operation limits of a present vehicle cruise control. As reported in the works of [5], simulated path following proved to achieve the smallest error when a range of 80 to 120 preview points are applied. Therefore, the same range of preview points which allows good accuracy and precision is being employed.

4.2.1 Obstacle Course (Straight Path)

Figure 4.2 depicts the vehicle trajectory as to avoid obstacles on the obstacle course, an intended straight path before obstacles are included. The selected path generated by the obstacle collision avoidance system described and shown earlier in Figure 4.1 is fed into the Neural Network path following/lane keeping system of the vehicle to accurately and precisely follow the generated path.

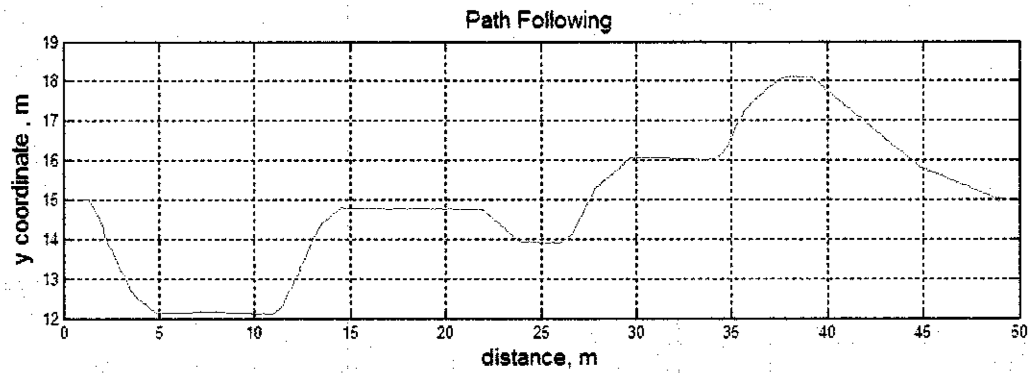


Figure 4.2: Vehicle path on obstacle course (straight path) for path following/lane keeping

4.2.1.1 Obstacle Course (Straight Path) at 80 km/h

The path following is as shown in Figure 4.2 and the vehicle traveling at a speed of 80 km/h is examined. Initially, when preview points of 80 are applied, the average Y path following error is 0.0106 m (10.6000 mm). By applying preview points of 120, the Neural Network converged to a desired output with an average Y path following error of 0.0090 m (9.0000 mm). As tabulated in Table 4.1, the average Y path following error reduces with the increasing of preview points although other parameters such as clockwise and counter clockwise steering wheel angle are almost identical. The results and observations for the obstacle course (straight path) at 80 km/h are presented in the following Table 4.1 and Figure 4.3.

Table 4.1: Summary of obstacle course (straight path) at 80 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
80	80	0.0106	0.3436	0.7923
	90	0.0100	0.3436	0.7924
	100	0.0095	0.3436	0.7924
	110	0.0091	0.3436	0.7924
	120	0.0090	0.3436	0.7924

Figure 4.3: Obstacle course (straight path) at 80 km/h with 120 preview points

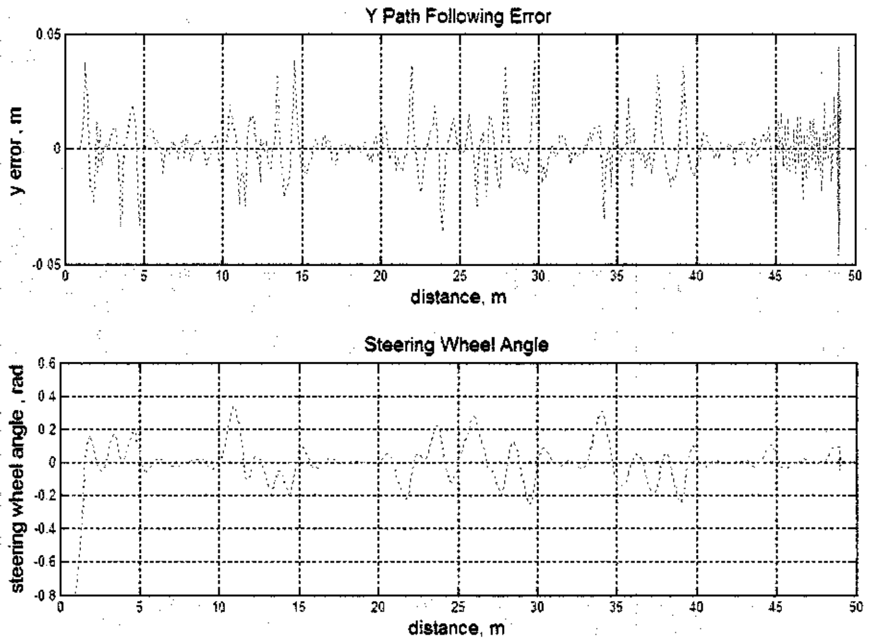


Figure 4.3(i) – (top): Y path following error

Figure 4.3(ii) – (bottom): Steering wheel angle

4.2.1.2 Obstacle Course (Straight Path) at 90 km/h

For the speed of 90 km/h which is the speed limit of country roads or trunk roads and certain stretches of expressways or highways, the simulation shows that 120 preview points displayed excellent result with the lowest average Y path following error as compared to the other preview points. A speed of 90 km/h with 120 preview points yielded an average Y path following error of 0.0089 m (8.9000 mm), a clockwise and counter clockwise steering wheel angle of 0.2988 rad and 0.7656 rad respectively. The results and observations for the obstacle course (straight path) at 90 km/h are presented in the following Table 4.2 and Figure 4.4.

Table 4.2: Summary of obstacle course (straight path) at 90 km/h

Speed, km/h	**Preview Points**	**Average Y Path Following Error, m**	**Steering Wheel Angle, rad**	
			CW	**CCW**
90	80	0.0105	0.2988	0.7656
	90	0.0099	0.2988	0.7656
	100	0.0094	0.2988	0.7656
	110	0.0090	0.2988	0.7656
	120	0.0089	0.2988	0.7656

Figure 4.4: Obstacle course (straight path) at 90 km/h with 120 preview points

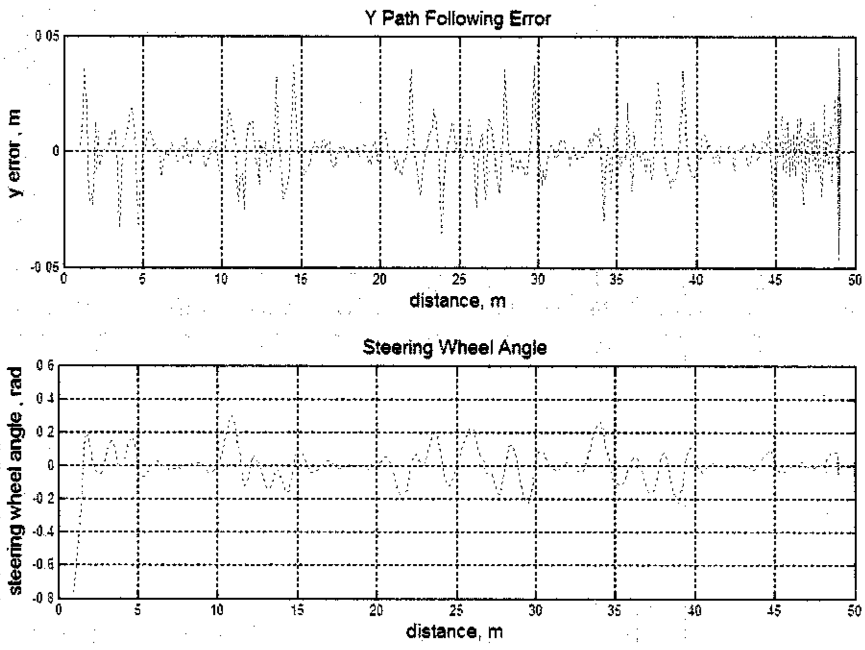


Figure 4.4(i) – (top): Y path following error

Figure 4.4(ii) – (bottom): Steering wheel angle

4.2.1.3 Obstacle Course (Straight Path) at 100 km/h

The vehicle traveling at 100 km/h proved capable of following the path designed to avoid the obstacles on the obstacle course (straight path) with much accuracy and precision as compared to the lower speed examined previously. 120 preview points proved to yield the lowest average Y path following errors judging by the other preview points. The Neural Network controller of the vehicle converged to the desired output with an average Y path following error of 0.0088 m (8.8000 mm) when a 120 preview points are applied. Clockwise and counter clockwise steering wheel angle produced indistinguishable results with the increasing of preview points.

The results and observations for the obstacle course (straight path) at 100 km/h are presented in the following Table 4.3 and Figure 4.5.

Table 4.3: Summary of obstacle course (straight path) at 100 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
100	80	0.0104	0.2638	0.7481
	90	0.0098	0.2638	0.7481
	100	0.0093	0.2638	0.7481
	110	0.0089	0.2638	0.7481
	120	0.0088	0.2638	0.7481

Figure 4.5: Obstacle course (straight path) at 100 km/h with 120 preview points

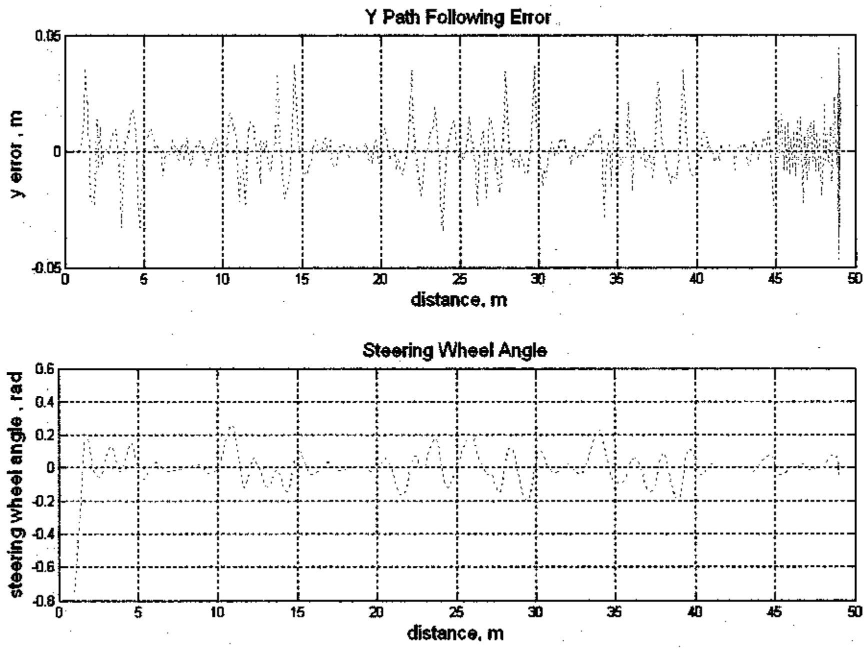


Figure 4.5(i) – (top): Y path following error

Figure 4.5(ii) – (bottom): Steering wheel angle

4.2.1.4 Obstacle Course (Straight Path) at 110 km/h

The speed is further increased to 110 km/h which simulates the speed limit on expressways and highways. As in the previous simulations, all parameters almost coincide except for the Y path following errors. Once again, 120 preview points appears to produce the lowest average Y path following error in comparison with other preview points. The vehicle traveling at 110 km/h manifests a high level of accuracy and precision in avoiding the obstacles on the obstacle course with an average Y path following error of 0.0088 m (8.8000 mm). In spite of the fact that the speed is increasing, the Y path following results of 110 km/h is an almost match to those of the lower speeds. The results and observations for the obstacle course (straight path) at 110 km/h are presented in the following Table 4.4 and Figure 4.6.

Table 4.4: Summary of obstacle course (straight path) at 110 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
110	80	0.0104	0.2386	0.7364
	90	0.0098	0.2387	0.7362
	100	0.0093	0.2387	0.7363
	110	0.0089	0.2387	0.7363
	120	0.0088	0.2387	0.7363

Figure 4.6: Obstacle course (straight path) at 110 km/h with 120 preview points

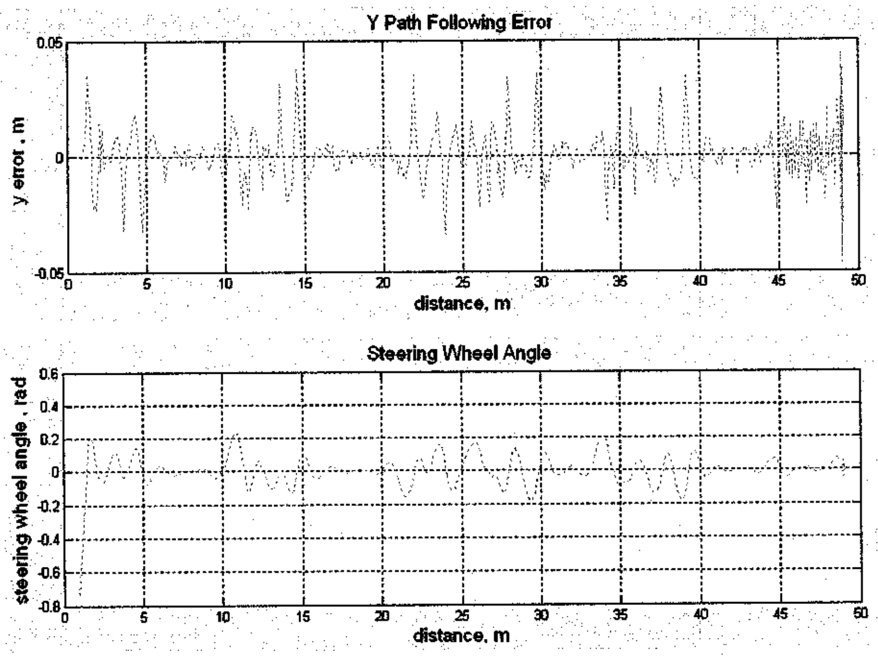


Figure 4.6(i) – (top): Y path following error

Figure 4.6(ii) – (bottom): Steering wheel angle

4.2.1.5 Obstacle Course (Straight Path) at 120 km/h

The results and observations for the obstacle course (straight path) at 120 km/h are presented in the following Table 4.5 and Figure 4.7 in Appendix A.

Table 4.5: Summary of obstacle course (straight path) at 120 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
120	80	0.0103	0.2222	0.7284
	90	0.0098	0.2224	0.7281
	100	0.0092	0.2223	0.7282
	110	0.0088	0.2223	0.7282
	120	0.0087	0.2223	0.7282

4.2.1.6 Obstacle Course (Straight Path) at 130 km/h

The results and observations for the obstacle course (straight path) at 130 km/h are presented in the following Table 4.6 and Figure 4.8 in Appendix A.

Table 4.6: Summary of obstacle course (straight path) at 130 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
130	80	0.0103	0.2322	0.7230
	90	0.0098	0.2325	0.7226
	100	0.0092	0.2324	0.7227
	110	0.0088	0.2324	0.7227
	120	0.0087	0.2324	0.7227

4.2.1.7 Obstacle Course (Straight Path) at 140 km/h

The results and observations for the obstacle course (straight path) at 140 km/h are presented in the following Table 4.7 and Figure 4.9 in Appendix A.

Table 4.7: Summary of obstacle course (straight path) at 140 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
140	80	0.0103	0.2385	0.7192
	90	0.0098	0.2389	0.7187
	100	0.0092	0.2388	0.7188
	110	0.0088	0.2388	0.7188
	120	0.0087	0.2388	0.7188

4.2.1.8 Obstacle Course (Straight Path) at 150 km/h

The results and observations for the obstacle course (straight path) at 150 km/h are presented in the following Table 4.8 and Figure 4.10 in Appendix A.

Table 4.8: Summary of obstacle course (straight path) at 150 km/h

Speed, km/h	**Preview Points**	**Average Y Path Following Error, m**	**Steering Wheel Angle, rad**	
			CW	**CCW**
150	80	0.0103	0.2488	0.7167
	90	0.0097	0.2492	0.7161
	100	0.0092	0.2491	0.7162
	110	0.0088	0.2491	0.7162
	120	0.0087	0.2491	0.7162

4.2.1.9 Obstacle Course (Straight Path) at 160 km/h

The results and observations for the obstacle course (straight path) at 160 km/h are presented in the following Table 4.9 and Figure 4.11 in Appendix A.

Table 4.9: Summary of obstacle course (straight path) at 160 km/h

Speed, km/h	**Preview Points**	**Average Y Path Following Error, m**	**Steering Wheel Angle, rad**	
			CW	**CCW**
160	80	0.0103	0.2575	0.7149
	90	0.0097	0.2580	0.7143
	100	0.0092	0.2579	0.7144
	110	0.0088	0.2579	0.7144
	120	0.0087	0.2579	0.7144

4.2.1.10 Obstacle Course (Straight Path) at 170 km/h

The results and observations for the obstacle course (straight path) at 170 km/h are presented in the following Table 4.10 and Figure 4.12 in Appendix A.

Table 4.10: Summary of obstacle course (straight path) at 170 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
170	80	0.0103	0.2641	0.7138
	90	0.0097	0.2647	0.7131
	100	0.0092	0.2646	0.7132
	110	0.0088	0.2646	0.7132
	120	0.0087	0.2646	0.7132

4.2.1.11 Obstacle Course (Straight Path) at 180 km/h

The results and observations for the obstacle course (straight path) at 180 km/h are presented in the following Table 4.11 and Figure 4.13 in Appendix A.

Table 4.11: Summary of obstacle course (straight path) at 180 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
180	80	0.0103	0.2691	0.7132
	90	0.0097	0.2698	0.7124
	100	0.0092	0.2697	0.7125
	110	0.0088	0.2697	0.7125
	120	0.0087	0.2697	0.7125

4.2.1.12 Obstacle Course (Straight Path) at 190 km/h

The results and observations for the obstacle course (straight path) at 190 km/h are presented in the following Table 4.12 and Figure 4.14 in Appendix A.

Table 4.12: Summary of obstacle course (straight path) at 190 km/h

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
190	80	0.0103	0.2729	0.7128
	90	0.0098	0.2737	0.7120
	100	0.0092	0.2736	0.7121
	110	0.0088	0.2735	0.7121
	120	0.0087	0.2736	0.7121

4.2.1.13 Obstacle Course (Straight Path) at 200 km/h

For the ultimate speed of 200 km/h, Table 4.13 attests to the vehicle's ability to follow the path designed to avoid obstacles. At such high speed, the Neural Network controller yet again proved to converge to the targeted output with an average Y path following error of 0.0087 m (8.7000 mm). Clockwise counter clockwise steering wheel angle is 0.2764 rad and 0.7120 rad respectively with 120 preview points applied. Clockwise steering wheel angle is somewhat decreasing with the increment of preview points. The results and observations for the obstacle course (straight path) at 200 km/h are presented in the following Table 4.13 and Figure 4.15.

Table 4.13: Summary of obstacle course (straight path) at 200 km/h

Speed, km/h	**Preview Points**	**Average Y Path Following Error, m**	**Steering Wheel Angle, rad**	
			CW	**CCW**
200	80	0.0103	0.2757	0.7127
	90	0.0098	0.2766	0.7118
	100	0.0092	0.2765	0.7120
	110	0.0088	0.2764	0.7120
	120	0.0087	0.2764	0.7120

Figure 4.15: Obstacle course (straight path) at 200 km/h with 120 preview points

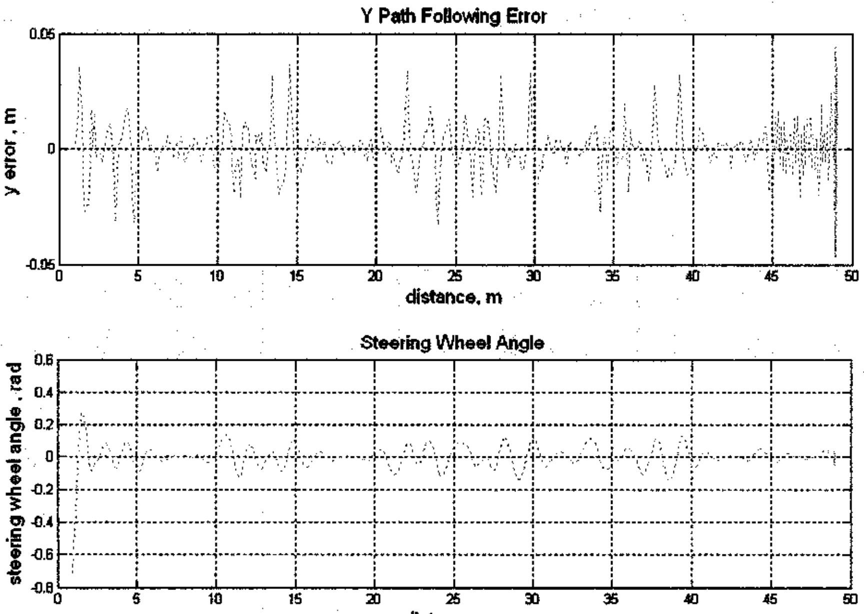


Figure 4.15(i) – (top): Y path following error

Figure 4.15(ii) – (bottom): Steering wheel angle

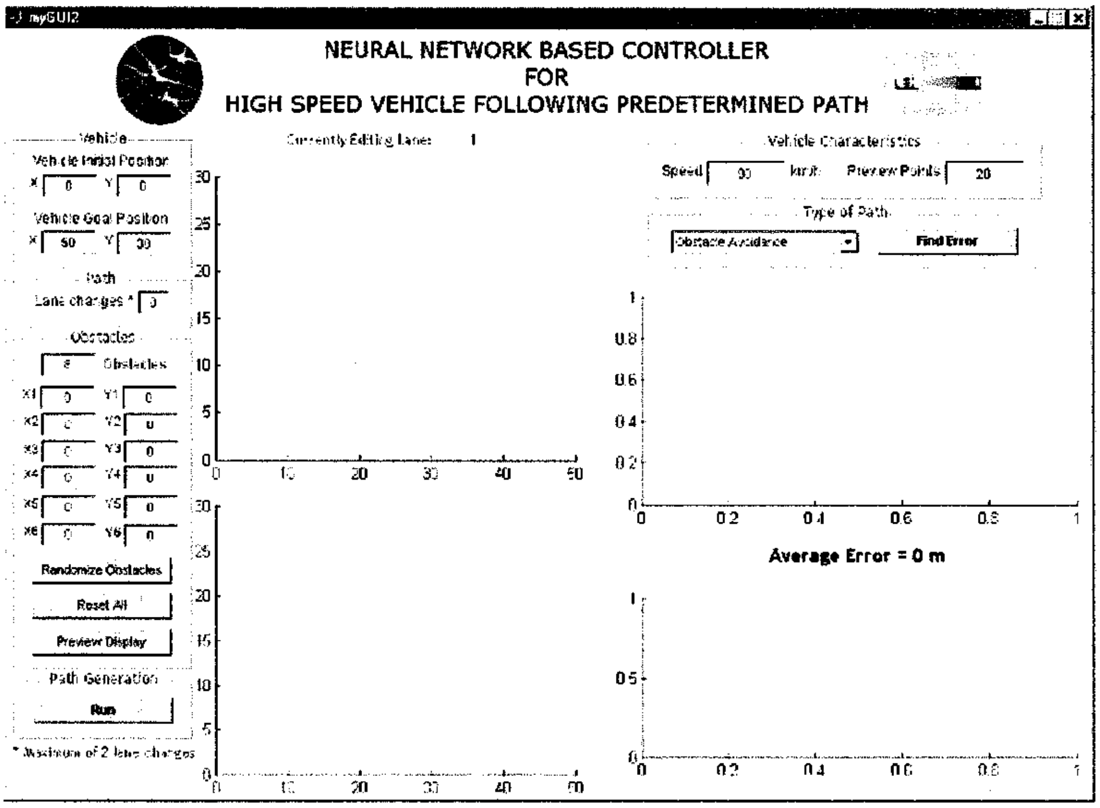
All path following simulations on the obstacle course (straight path) give excellent results. The average Y path following errors obtained is very small. They vary from 0.0087 m (8.7000 mm) to 0.0106 m (10.6000 mm). The average Y path following errors depends on many parameters such as: the number of preview points used (or similarly the time ahead of the car taken into account in the optimal controller), the priorities chosen for the optimal controller, the car parameters (mass and the yaw inertia especially), the speed of the car and the path chosen [12]. The average Y path following error decreases with the increase of preview points as one can expect. By choosing a high number of preview points and by setting high priorities on path following, it is possible to obtain lesser Y path following error. It is also worth noticing that the Y path following error and steering wheel angle fluctuates less when traveling on a straight section of the path. On the contrary, the vehicle produces a larger Y path following error and requires a larger magnitude of steering wheel angle when negotiating a bend, curve or turn.

4.2.2 Mobile Obstacles and Uncertainty

The assumption of stationary obstacles will not hold in any real obstacle avoidance problem. Stationary obstacles can be represented as pot-holes, stalled vehicle or objects of any kind (debris, carcass or fallen branch of a tree) encountered along the path. If the obstacles environment is dynamic in the sense that, for instance, obstacles can move such as another vehicle up-front, this system requires extensions. For instance if some obstacles suddenly appeared at some position and the vehicle did not "know" about it, it can simply collide with it. Or if the obstacles moved in predictable ways, it should be clear that a "look-ahead strategy" may be needed. If the obstacles moved in unpredictable ways, then the model may not be able to accurately represent this so the vehicle will need to sense the environment while it navigates it and try to learn about the obstacles positions and movements.

4.3 Graphical User Interface (GUI)

To make it easier to simulate path following/lane keeping with obstacle avoidance system of the vehicle, the aim was to build a Graphical User Interface (GUI) with all useful possibilities. The GUI is designed to be simple and user friendly. The GUI as shown in Figure 4.16 is divided into six panels for some settings and six displays for some information.


myGUI2

NEURAL NETWORK BASED CONTROLLER
FOR
HIGH SPEED VEHICLE FOLLOWING PREDETERMINED PATH

Vehicle
Vehicle Initial Position X 0 Y 0
Vehicle Goal Position X 50 Y 30

Path
Lane changes * 0

Obstacles
6 Obstacles
X1 0 Y1 0
X2 0 Y2 0
X3 0 Y3 0
X4 0 Y4 0
X5 0 Y5 0
X6 0 Y6 0

Path Generation

* Maximum of 2 lane changes

Currently Editing Lane: 1

Vehicle Characteristics
Speed 90 km/h Preview Points 20

Type of Path
Obstacle Avoidance

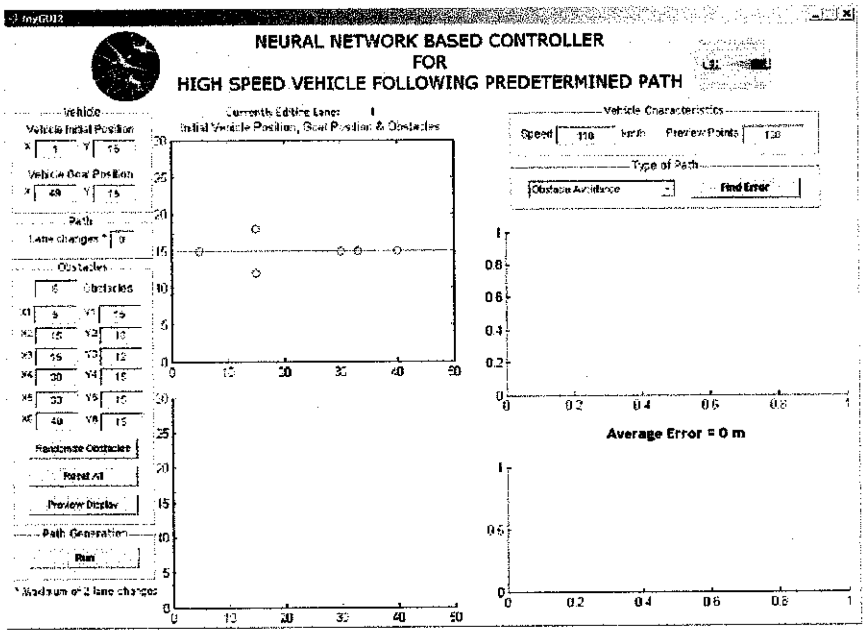
Average Error = 0 m

Figure 4.16: Graphical User Interface (GUI) screenshot

4.3.1 Obstacle Course GUI Screenshot

4.3.1.1 Straight Path

The first panel is "Vehicle". In this panel vehicle initial and goal position can be specified but the test field for the vehicle to travel within is constrained to [50, 30]. The second panel "Path" is for the lane change setup. Setting "0" is for the vehicle to travel in a straight path (from one end to another), "1" is for sudden change of direction and "2" is for lane change. In the third panel "Obstacles", the number of obstacles and the locations of the obstacles can be varied. Maximum of 6 obstacles can be selected and each obstacle position must comply with the test field boundaries which is limited to [50, 30]. At the same time, the "Randomize" push button allows the positions of the obstacles to be randomized automatically. The "Preview" push button enables the user to preview the path of the vehicle and also the location of the obstacles as shown in Figure 4.17. The user is able to edit any information after the preview or reset all the information previously stored with the "Reset All" push button. "Run" push button on the "Path Generation" panel generates the path of the vehicle as to avoid the obstacles and stay on course to the goal as depicted in Figure 4.18.


myGUI2

NEURAL NETWORK BASED CONTROLLER
FOR
HIGH SPEED VEHICLE FOLLOWING PREDETERMINED PATH

Vehicle
Vehicle Initial Position
X 1 Y 15
Vehicle Goal Position
X 49 Y 15

Path
Lane changes * 0

Obstacles
6 Obstacles
X1 5 Y1 15
X2 15 Y2 18
X3 15 Y3 12
X4 30 Y4 15
X5 33 Y5 15
X6 40 Y6 15

Path Generation

* Maximum of 2 lane changes

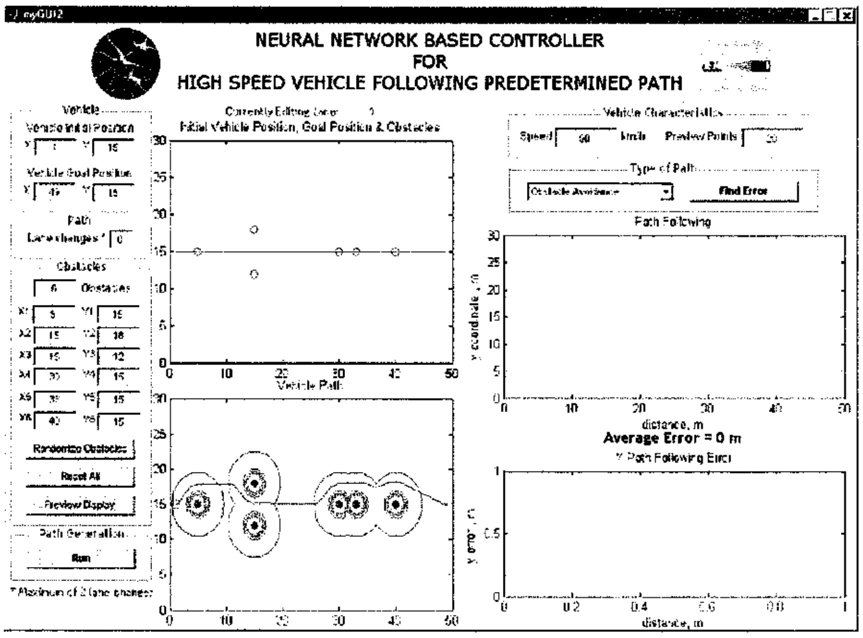
Currently Editing Lane: 1
Initial Vehicle Position, Goal Position & Obstacles

Vehicle Characteristics
Speed 110 km/h Preview Points 130

Type of Path
Obstacle Avoidance

Average Error = 0 m

Figure 4.17: GUI screenshot of path of vehicle and positions of obstacles preview



Vehicle

Vehicle Initial Position: X 1, Y 15

Vehicle Goal Position: X 49, Y 15

Path: Lane changes * 0

Obstacles: 6 Obstacles

X1	5	Y1	15
X2	15	Y2	18
X3	15	Y3	12
X4	30	Y4	15
X5	33	Y5	15
X6	40	Y6	15

Path Generation

* Maximum of 2 lane changes

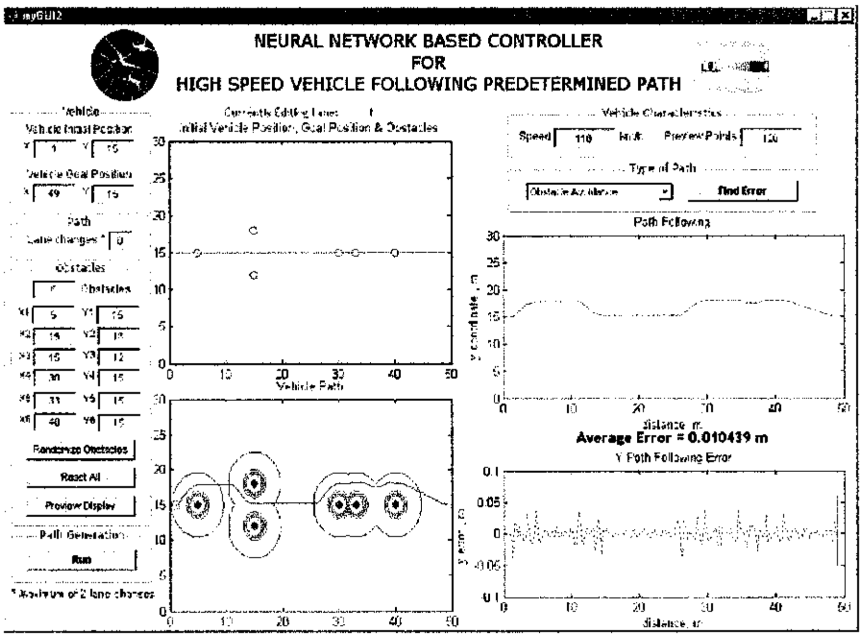
Vehicle Characteristics: Speed 60 km/h, Preview Points 20

Type of Path: Obstacle Avoidance

Average Error = 0 m

Figure 4.18: GUI screenshot of path of the vehicle as to avoid the obstacles and stay on course to the goal

After obtaining the path of vehicle designed to avoid the obstacles, the ability of the vehicle to accurately and precisely follow the path can be gauged. The speed and preview points of vehicle are determined foremost in the "Vehicle Characteristics" panel. In the next panel "Type of Path" there are five pop up menus viz. "Obstacle Avoidance", "Sinus Shape", "Lane Change", "Sudden Change of Direction" and "Smooth Random Path". There the user can select the appropriate type of path where the Neural Network can learn to converge to the targeted output. "Obstacle Avoidance" option is selected for an obstacle course which is limited to straight path, lane change and sudden change of direction as mentioned earlier, whereas the rest are designed to perform for an obstacle free path. Also on this panel, is "Find Error" push button to calculate the performance of the Neural Network which is the average Y path following error of the vehicle. After the processing of the Neural Network, simulation results of the path followed and the Y path following error jointly with the average Y path following error value are displayed. The GUI screenshot for obstacle course (straight path) at 110 km/h with 120 preview points is depicted in Figure 4.19.

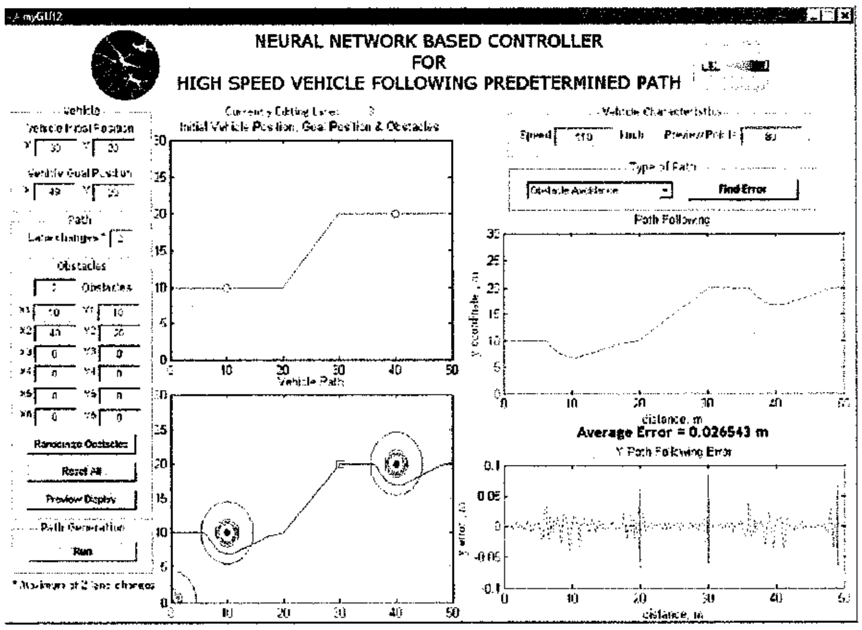


Vehicle Initial Position X 1 Y 15
Vehicle Goal Position X 49 Y 15
Lane changes * 0
Obstacles 6
X1 5 Y1 15
X2 15 Y2 18
X3 15 Y3 12
X4 30 Y4 15
X5 33 Y5 15
X6 40 Y6 15
Speed 110 km/h Preview Points 120
Type of Path: Obstacle Avoidance
Average Error = 0.010439 m

Figure 4.19: GUI screenshot of path followed and Y path following error of the vehicle

4.3.1.2 Lane Change

The GUI screenshot for lane change with obstacles at 110 km/h with 80 preview points is depicted in Figure 4.20.

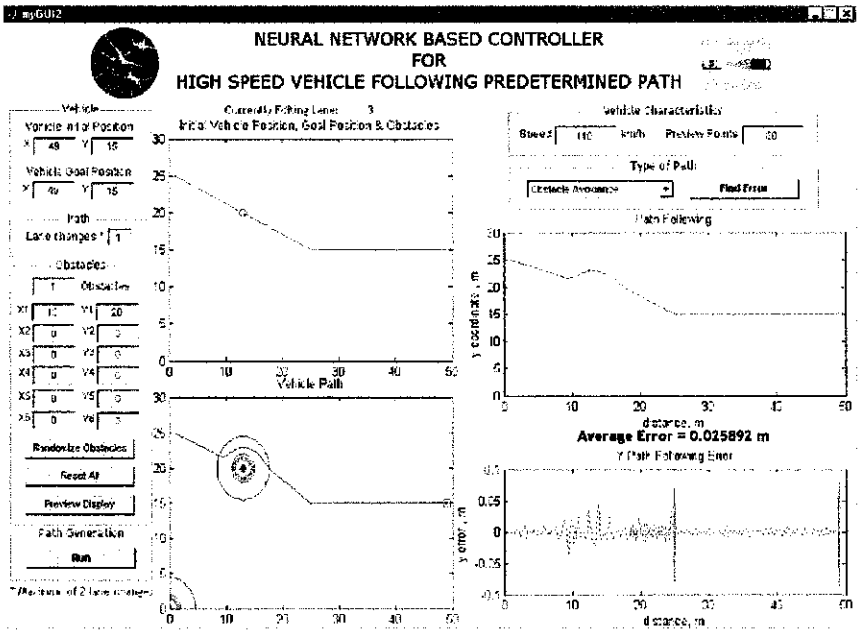


Currently Editing Lane: 3
Vehicle Initial Position X 1 Y 10
Vehicle Goal Position X 49 Y 20
Lane changes * 2
Obstacles 2
X1 10 Y1 10
X2 40 Y2 20
X3 0 Y3 0
X4 0 Y4 0
X5 0 Y5 0
X6 0 Y6 0
Speed 110 km/h Preview Points 80
Type of Path: Obstacle Avoidance
Average Error = 0.026543 m

Figure 4.20: GUI screenshot of lane change with obstacles at 110 km/h with 80 preview points

4.3.1.3 Sudden Change of Direction

The GUI screenshot for sudden change of direction with obstacles at 110 km/h with 80 preview points is depicted in Figure 4.21.



myGUI2

NEURAL NETWORK BASED CONTROLLER
FOR
HIGH SPEED VEHICLE FOLLOWING PREDETERMINED PATH

Vehicle

Vehicle Initial Position X 49 Y 15

Vehicle Goal Position X 49 Y 15

Path

Lane changes * 1

Obstacles

1 Obstacles

X1 10 Y1 20

X2 0 Y2 0

X3 0 Y3 0

X4 0 Y4 0

X5 0 Y5 0

X6 0 Y6 0

Currently Editing Lane: 3

Vehicle Characteristics

Speed 110 km/h Preview Points 80

Type of Path: Obstacle Avoidance

Average Error = 0.025892 m

*Maximum of 2 lane changes

Figure 4.21: GUI screenshot of sudden change of direction with obstacles at 110 km/h with 80 preview points

4.3.2 Predetermined Obstacle Free Paths GUI Screenshot

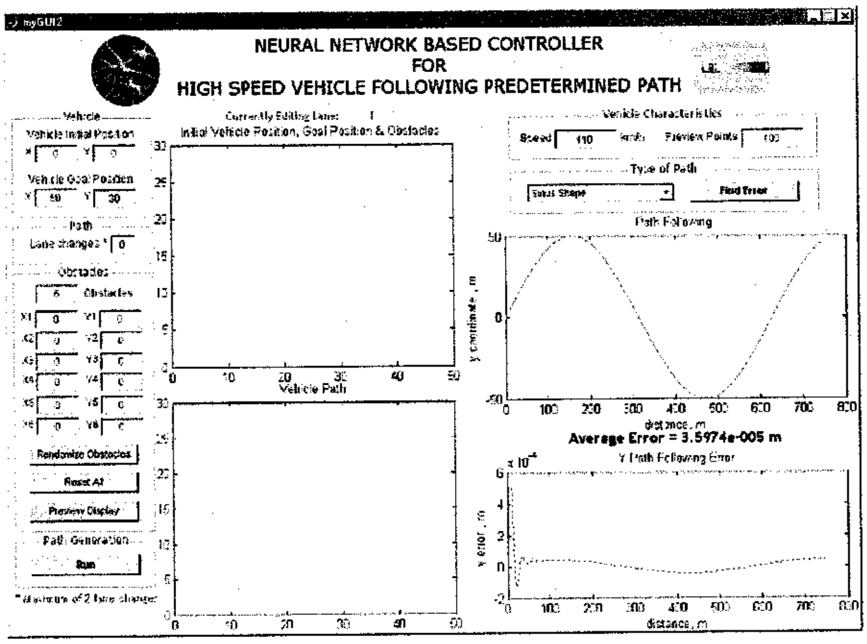
This section is dedicated to simulate path following on an assortment of predetermined paths without obstacles in GUI. In order to simulate a wide range of circuits, several road sections created in the works of [12] have been used here. These are the different road sections selected to represent most maneuvers. In works by Baharudin [5], four different paths: sinus path, lane change, sudden change of direction and smooth random path were simulated with the increasing speed of vehicle as well as preview points. It is worth noticing that all road sections are feasible maneuvers with respect to the speed chosen (80 km/h to 110 km/h). As reported in the works of [5], the path following simulation of these paths proved to achieve the smallest error when a range of 80 to 120 preview points are applied. In this section, the path following ability of the vehicle on four different obstacle free paths is examined once again but in the GUI developed in this thesis. The median value of 100 preview points is being employed on the vehicle speed limit on expressway, 110 km/h.

4.3.2.1 Sinus Path

The succession of curves consisting of a cosine shaped road is as shown in Figure 4.22. The simulation results in Table 4.14 shows that 110 km/h with 100 preview points produces an astounding average Y path of 3.5974×10^{-5} m (0.035970 mm). Clockwise and counter clockwise steering wheel angle are both 0.1962 rad. The Neural Network is much more accurate in sinus path, yielding average Y path following errors of less than 10^{-5} m as well as clockwise and counter clockwise steering wheel angle of the same magnitude, because the turns are smooth as the road input is a sinus at a specific frequency and it is not surprising to obtain better results on a sinus path. The tabulated results and observation as well as the GUI screenshot for the sinus path at 110 km/h with 100 preview points are presented in the following Table 4.14 and Figure 4.22.

Table 4.14: Summary of sinus path without obstacles at 110 km/h with 100 preview points

Speed, km/h	**Preview Points**	**Average Y Path Following Error, m**	**Steering Wheel Angle, rad**	
			CW	**CCW**
110	100	3.5974×10^{-5}	0.1962	0.1962


myGUI2

NEURAL NETWORK BASED CONTROLLER
FOR
HIGH SPEED VEHICLE FOLLOWING PREDETERMINED PATH

Vehicle
Vehicle Initial Position: X 0, Y 0
Vehicle Goal Position: X 50, Y 30

Path
Lane changes * 0

Obstacles
6 Obstacles
X1 0 Y1 0
X2 0 Y2 0
X3 0 Y3 0
X4 0 Y4 0
X5 0 Y5 0
X6 0 Y6 0

Path Generation

* Maximum of 2 lane changes

Currently Editing Lane: 1

Vehicle Characteristics
Speed 110 km/h Preview Points 100

Type of Path
Sinus Shape

Average Error = 3.5974e-005 m

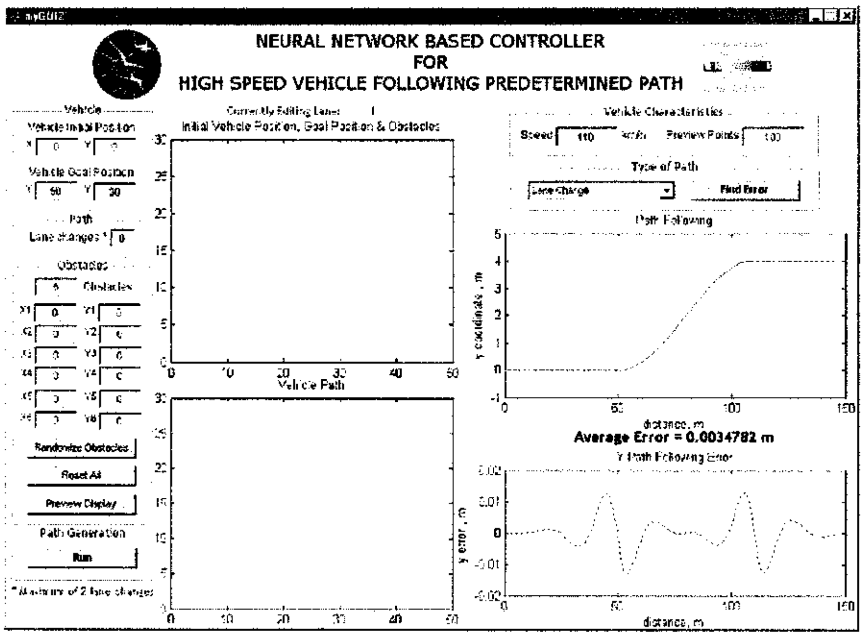
Figure 4.22: GUI screenshot of sinus path without obstacles at 110 km/h with 100 preview points

4.3.2.2 Lane Change

The lane change consisting of a cosine shaped lateral shift joining two straight lines is as shown in Figure 4.23. From the result of lane change simulation tabulated in Table 4.15, an average Y path following error of 0.0035 m (3.5000 mm) is obtained. The tremendous difference of average Y path following error is due to the fact that even the change of direction in lane change is smooth but the turns are tight whereas in sinus path, road input is a sinus at a specific frequency. The tabulated results and observation as well as the GUI screenshot for the lane change at 110 km/h with 100 preview points are presented in the following Table 4.15 and Figure 4.23.

Table 4.15: Summary of lane change without obstacles at 110 km/h with 100 preview points

Speed, km/h	**Preview Points**	**Average Y Path Following Error, m**	**Steering Wheel Angle, rad**	
			CW	**CCW**
110	100	0.0035	0.3271	0.2738


NEURAL NETWORK BASED CONTROLLER
FOR
HIGH SPEED VEHICLE FOLLOWING PREDETERMINED PATH

Vehicle Initial Position: X 0, Y 0
Vehicle Goal Position: X 50, Y 30
Lane changes* 0
Obstacles: 6
X1 0, Y1 0; X2 0, Y2 0; X3 0, Y3 0; X4 0, Y4 0; X5 0, Y5 0; X6 0, Y6 0
Currently Editing Lane: 1
Speed 110 km/h, Preview Points 100
Type of Path: Lane Change
Average Error = 0.0034782 m
*Maximum of 2 lane changes

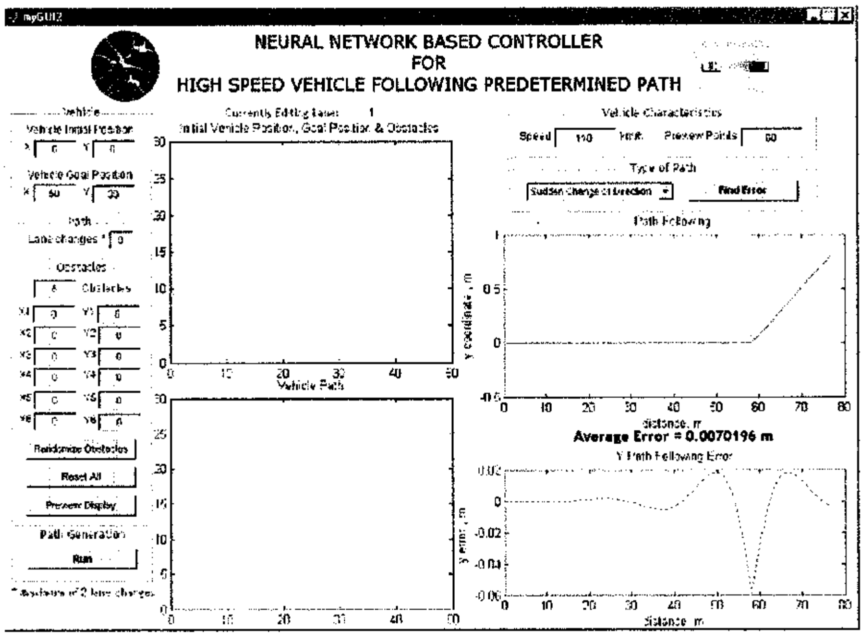
Figure 4.23: GUI screenshot of lane change without obstacles at 110 km/h with 100 preview points

4.3.2.3 Sudden Change of Direction

The sudden change of direction is as shown in Figure 4.24. Referring to the works of [5], it has been reported that a maximum preview points of 80 is allowable to achieve the targeted result due to the fact that sudden change contains high frequencies and implies that the tracked path is closer to random noise. However in this simulation, the vehicle speed remains as 110 km/h. Table 4.16 displays the path following ability of the vehicle traveling at 110 km/h generating an average Y path following error of 0.0070 m (7.0000 mm). Clockwise and counter clockwise steering wheel angle appears to be 0.2649 rad and 0.1731 rad respectively. The tabulated results and observation as well as the GUI screenshot for the sudden change of direction at 110 km/h with 80 preview points are presented in the following Table 4.16 and Figure 4.24.

Table 4.16: Summary of sudden change of direction without obstacles at 110 km/h with 80 preview points

Speed, km/h	Preview Points	Average Y Path Following Error, m	Steering Wheel Angle, rad	
			CW	CCW
110	80	0.0070	0.2649	0.1731


NEURAL NETWORK BASED CONTROLLER
FOR
HIGH SPEED VEHICLE FOLLOWING PREDETERMINED PATH

Vehicle Initial Position X 0 Y 0
Vehicle Goal Position X 50 Y 30
Lane changes * 0
Obstacles 6 Obstacles
X1 0 Y1 0
X2 0 Y2 0
X3 0 Y3 0
X4 0 Y4 0
X5 0 Y5 0
X6 0 Y6 0
Currently Editing Lane: 1
Vehicle Characteristics
Speed 110 km/h Preview Points 80
Type of Path: Sudden Change of Direction
Average Error = 0.0070196 m
* maximum of 2 lane changes

Figure 4.24: GUI screenshot of sudden change of direction without obstacles at 110 km/h with 80 preview points

4.3.2.4 Smooth Random Path

The smooth random path is as shown in Figure 4.25. The vehicle traveling at 110 km/h with 100 preview points manifests a high level of accuracy and precision in following the path with an average Y path error of 1.3684x10^{-5} m (0.013684 mm). Both smooth random path and sinus path appears to produce outstanding results in terms of average Y path following errors with less than 10^{-5} m. The fact that it is possible to obtain better results with smooth random path than with obstacle course, lane change or sudden change of direction is not surprising. Since the random path has been smoothed, most high frequencies have disappeared and the random path is not representative of a noise disturbance. The tabulated results and observation as well as the GUI screenshot for the smooth random path at 110 km/h with 100 preview points are presented in the following Table 4.17 and Figure 4.25.

Table 4.17: Summary of smooth random path without obstacles at 110 km/h with 100 preview points

Speed, km/h	Preview Points	Average Y Path Following Error, m	Max Steering Wheel Angle, °	Min Steering Wheel Angle, °
110	100	1.3684×10^{-5}	4.658147	-5.72958

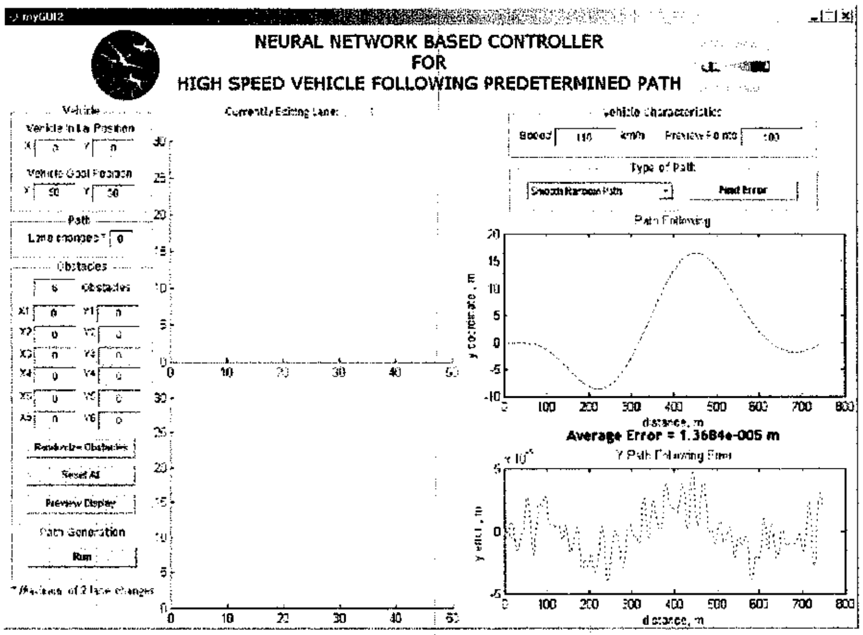


Figure 4.25: GUI screenshot of smooth random path without obstacles at 110 km/h with 100 preview points

All path following simulations for four different obstacle free paths: sinus path, lane change, sudden change of direction and smooth random path give excellent results. The Y path following errors obtained after is very small for all four paths. They vary from 3.5974×10^{-5} m (0.035970 mm) for the sinus path to 0.0070 m (7.0000 mm) for the sudden change of direction. The Y path following errors depends on many parameters such as: the number of preview points used (or similarly the time ahead of the car taken into account in the optimal controller), the priorities chosen for the optimal controller, the car parameters (mass and the yaw inertia especially), the speed of the car and the path chosen [12]. By choosing a high number of preview points and by setting high priorities on path following, it is possible to obtain Y path following errors of less than 10^{-6} m. In [12], the Y path following errors of sinus road are tracked for several speeds as a function of the time ahead used in the optimal controller. The minimum time required to track the path increases with the speed as one can expect. Preview gains study by Dandré [12] has shown the importance of choosing a suitable preview points to be considered in the road preview model. Study by Dandré [12] highlighted also that even small preview gains can have a strong effect on the Y path following errors.

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

This thesis emphasized on vehicle guidance for obstacle collision avoidance operating conjointly with the works of [5], [12], [13] and [18] on path following/lane keeping. The MATLAB simulation using Gaussian functions and multi-objective cost function proved the capability of a vehicle to precisely maneuver and avoid collision with obstacles along its path by assuming stationary and static obstacles. The collaboration of path following/lane keeping system (utilizing Neural Network controller and optimal controller) with obstacle collision avoidance system (utilizing Gaussian functions and multi-objective cost function) performed substantially well in avoiding collision with obstacles while traveling on high speed along its predetermined path. Vehicle traveling at a range of speed of 80 km/h to 200 km/h on an obstacle course demonstrated its ability to avoid collision with obstacles whilst following the predetermined path with much precision and accuracy. The average Y path following errors obtained for obstacle course varying from 0.0087 m (8.7000 mm) to 0.0265 m (26.5000 mm) is very small and considered almost negligible. The Y path following errors depends on many parameters such as: the number of preview points used (or similarly the time ahead of the car taken into account in the optimal controller), the priorities chosen for the optimal controller, the car parameters (mass and the yaw inertia especially), the speed of the car and the path chosen [12]. By choosing a high number of preview points and by setting high priorities on path following, it is possible to achieve greater accuracy and precision.

This project has been very interesting and opened other possibilities for making cars safer. Nevertheless, we are now at the forefront of designing Smarter/Safer vehicles, when it comes to following a path/keeping on a lane and avoiding collision with obstacles while on high speed. We have hybrid-powered cars now, maybe one day soon hybrid (Human/Computer) controlled steering vehicles will be on the roads.

5.2 Proposal for Future Works

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

5.2.1 Employment of Model-Predictive Control (MPC) Method

No model is perfectly accurate and this will always be the case that the model will not be a perfect representation of the environment. This implies that there will always be uncertainty in planning and hence there will always be a bound on the amount of time that it makes sense to project (simulate the model) into the future. Projecting into the future too far becomes useless at some point since the predictions will become inaccurate at some point and hence provide no good information on how to select the best plan. The difficulty is knowing how good the model is and how far to project into the future. The employment of MPC will enable the vehicle to predict what goal is going to occur in the future since there could be a time-varying goal. If the goal can be predicted, contingencies can be developed and earlier plans may be modified to try to ensure success for not only the current goals, but anticipated ones.

5.2.2 Additional Neural Network Control of the Forward Speed

This additional feature will enable the vehicle to move in non-constant speed. This way, the network will reduce the velocity of the vehicle 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.

5.2.3 Implementation of Different Types of Path

At this time, simulation on obstacle course is only limited to straight path, lane change and sudden change of direction. In order to simulate a wide range of circuits, sinus path and smooth random path with obstacles can be included in future studies. Unexplored and unknown paths are also a substantive feature to be considered as the paths taken by a car varies constantly and it is impractical that a driver only travels along that narrow range of circuits. It would also be interesting to see the car model is able follow more challenging paths that have moving obstacles such as domestic animals or children crossing the road and moving light or heavy vehicle ahead of the car, just to name a few.

5.2.4 Improvement of Car Model

The car model can be improved to represent a genuine car in several manners: a suspension model with rolling motion can be defined; the right and left wheels on one axle can be decoupled considering load transfer and aerodynamic forces can be considered.

5.2.5 Implementation of Different Learning Process

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

5.2.6 Improvement of Neural Structure

The efficiency of the Neural Network can be improved by selecting a suitable structure of the network such as number of hidden layers and number of neurons per layer.

REFERENCE

[1] R. Mohamad Isa, *Global Road Safety Crisis*, The Plenary of the 58th Session of the General Assembly of the United Nations, New York, 14 April 2004, Agenda Item: 160.

[2] *Road Accident Causes Report 2003*, Traffic Division of Bukit Aman Police Headquarters, 2003.

[3] J. R. Treat, N. S. Tumbas, S. T. McDonald, D. Shinar, R. D. Hume, R. E. Mayer, R. L. Stanisfer and N. J. Castellan, *Tri-Level Study of the Causes of Traffic Accidents*, 1977, Report No. DOT-HS-034-3-535-77 (TAC).

[4] M. Green and J. Senders, 2004, *Human Error in Road Accidents*, http://www.visualexpert.com/Resources/roadaccidents.html

[5] N. H. Baharudin, *Neural Network Controller for Non-Linear Vehicle Following Predetermined Paths*, B.Sc. Thesis, Universiti Teknologi PETRONAS, December 2005.

[6] M. A. Goodrich and E. R. Boer, *Designing Human-Centered Automation Tradeoffs in Collision Avoidance System Design*, Proceedings of IEEE Transactions on Intelligent Transportation Systems, Vol. 1, No. 1, March 2000.

[7] P. Griffiths, D. Langer, J. A. Misener, M. Siegel and C. Thorpe, 2001, *Sensor-Friendly Vehicle and Roadway Systems*, Instrumentation and Measurement Technology Conference, 2001. (IMTC 2001), Proceedings of the 18th IEEE, Vol. 2, 21 – 23 May 2001, pp. 1036 – 1040.

[8] Riid, Andri, Pahhomov, Dmitri, Rustern and Ennu, 2004, *Car Navigation and Collision Avoidance with Fuzzy Logic*, Fuzzy Systems, 2004. Proceedings of IEEE International Conference 2004, Vol. 3, 25 – 29 July 2004, pp. 144 – 1448.

[9] K. C. Cheok, G. E. Smid and D. J. McCune, *A Multisensor Based Collision Avoidance System with Application to a Military HMMWV*, Proceedings of IEEE Intelligent Transportation Systems Conference 2000, Dearborn (MI), USA, 1 – 3 October 2000.

[10] Lages and Ulrich, *Collision Avoidance System for Fixed Obstacles – Fuzzy Controller Network for Robot Driving of an Autonomous Vehicle*, Proceedings of IEEE Intelligent Transportation Systems Conference 2001, Oakland (CA), USA, pp. 25 – 29, August 2001.

[11] Howells and C. Christopher, *Game-Theoretic Approach with Cost Manipulation to Vehicular Collision Avoidance*, Thesis, Virginia Polytechnic Institute and State University, 2004

[12] P. Dandré, *Learning Path Following Control of an Automobile*, M.Sc. Thesis, Imperial College London, September 2003.

[13] N. H. H. Mohamad Hanif, *Path Following Using a Learning Neural Network Controller*, M.Sc. Thesis, Imperial College London, September 2004.

[14] *Neural Network FAQ, Part 2 of 7: Learning*, http://www.accpc.com/nnfaq/FAQ2.html

[15] *Neural Network MATLAB Toolbox*, Version 6.5.

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

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

[18] K. M. Passino, *Biomimicry for Optimization, Control and Automation,* London: Springer-Verlag, 2005.

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

[20] R.S. Sharp, *Modelling of the handling problem.*

APPENDIX A

Tables

Table 1.1: Causes of fatal road accident on Malaysian roadways in 2003 (source: Traffic Division of Bukit Aman Police Headquarters, 2003)

Sebab-Sebab Kemalangan	Peratusan
Cuai melintas jalan (p/kaki)	24.64
Terbabas sendiri (satu kenderaan)	23.19
Cuba/sedang memotong/tukar lorong	18.84
Makan jalan (bukan memotong)	12.56
Cuai keluar/Masuk simpang jalan susur	7.26
Tidak nampak/perasan ada kenderaan dan lain-lain	6.76
Mengikut rapat	4.34
Pusing "U"/Patah balik/Menyeberang laluan	2.42
Melawan aliran trafik	1.93
Tidak ikut lampu isyarat merah	0.48
Berhenti mengejut	0.48
Leka/Berangan/Letih/Mengantuk	0.48
Mabuk/dadah	0
Cuai mengundur	0
Berlumba/Potong Zig-zag	0
Lain-lain	6.28
Tidak diketahui	4.83

Table 3.1: Vehicle model parameters (source: P. Dandré, 2003)

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)	120000 $Nrad^{-1}$
Cornering stiffness of rear axle tyres (C_r)	80000 $Nrad^{-1}$
Fixed Steering Ratio (Hand wheel/road wheel), G	17
Track Width, w	

Table 3.2: Vehicle model forces (source: P. Dandré, 2003)

Type of Forces	**Equations**
Front axle longitudinal force	$F_{xf} = F_{xfl} + F_{xfr}$
Front axle lateral force	$F_{yf} = F_{yfl} + F_{yfr}$
Rear axle longitudinal force	$F_{xr} = F_{xrl} + F_{xrr}$
Rear axle lateral force	$F_{yr} = F_{yrl} + F_{yrr}$

Figures

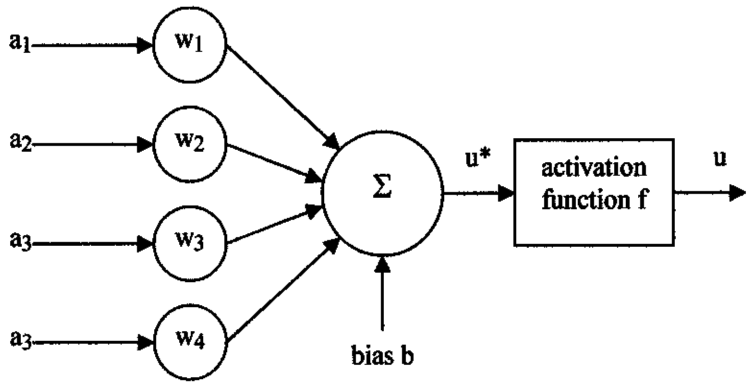


Figure 2.1: Model of a neuron (redrawn from P. Dandré, 2003)

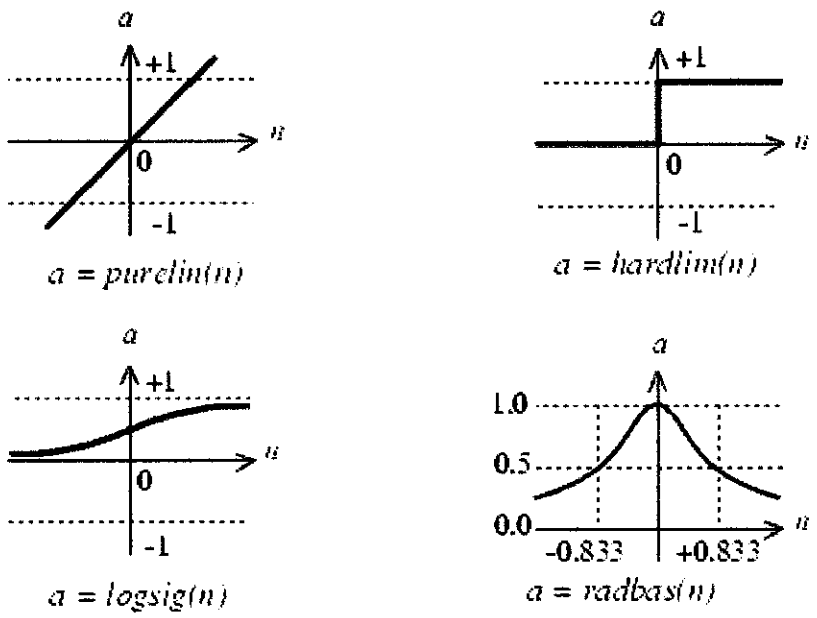


Figure 2.2: Four basic activation functions of a neuron (taken from P. Dandré, 2003)

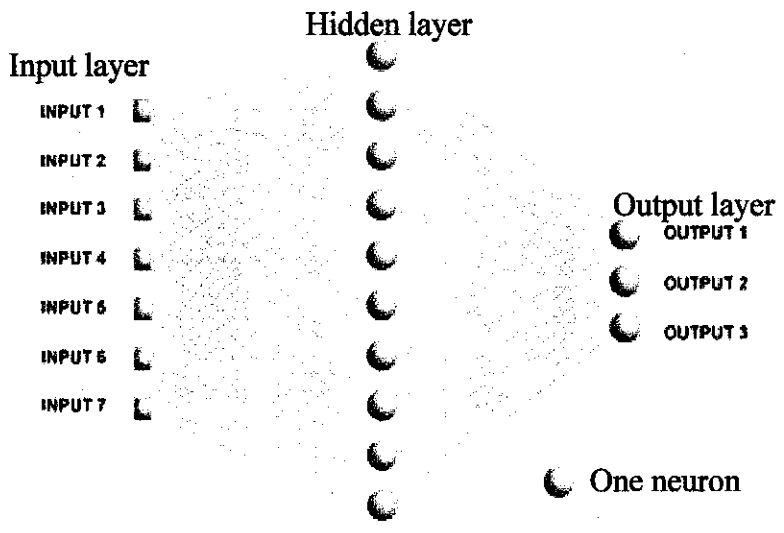


Figure 2.3: A basic multi-layer feed-forward network

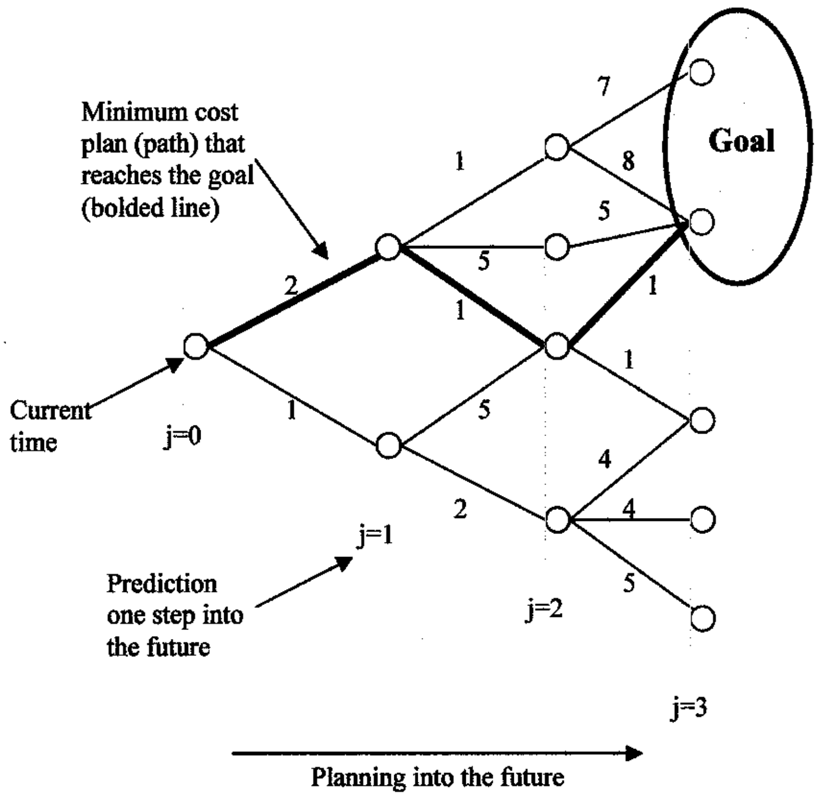

Minimum cost plan (path) that reaches the goal (bolded line)
Goal
7
8
5
1
5
2
1
1
1
1
5
4
4
2
5
Current time
j=0
j=1
j=2
j=3
Prediction one step into the future
Planning into the future

Figure 2.5: Tree representation of the alternative plans that can be considered at some point in time, along with the costs of executing such plans (redrawn from K. M. Passino, 2005)

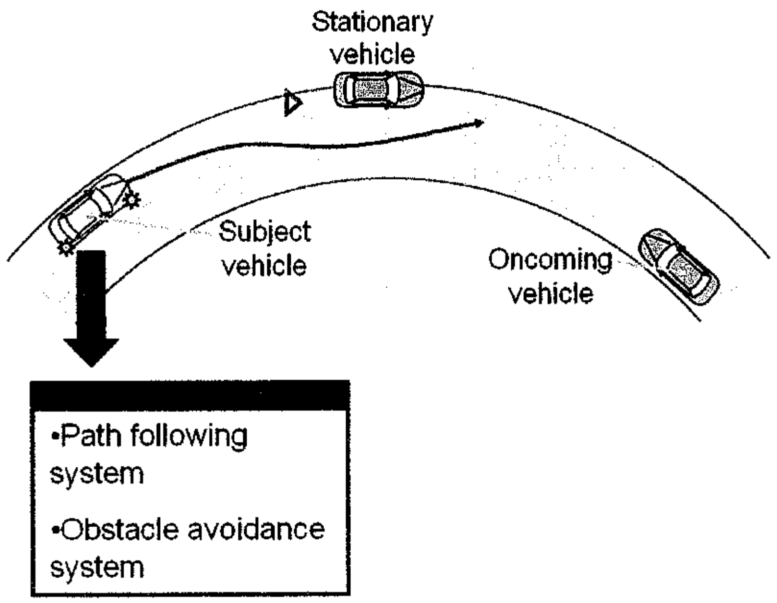


Figure 3.2: Vehicle steers away from another stationary vehicle or object appearing in the vehicle's path while traveling on a prescribed path/lane

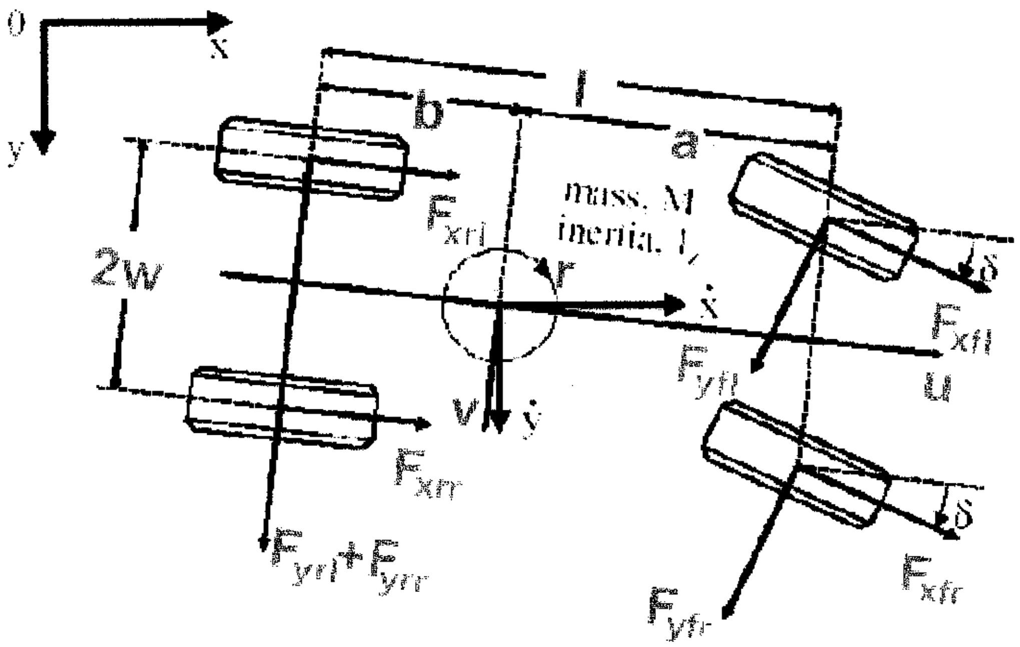


Figure 3.3: Plan view of the model of the car (taken from P. Dandré, 2003)

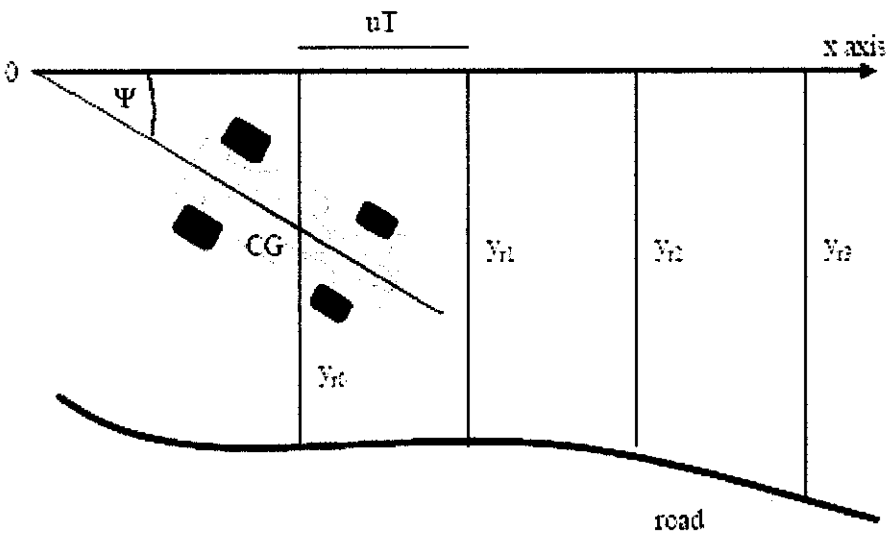


Figure 3.4: Car and Road at instant k (taken from P. Dandré, 2003)

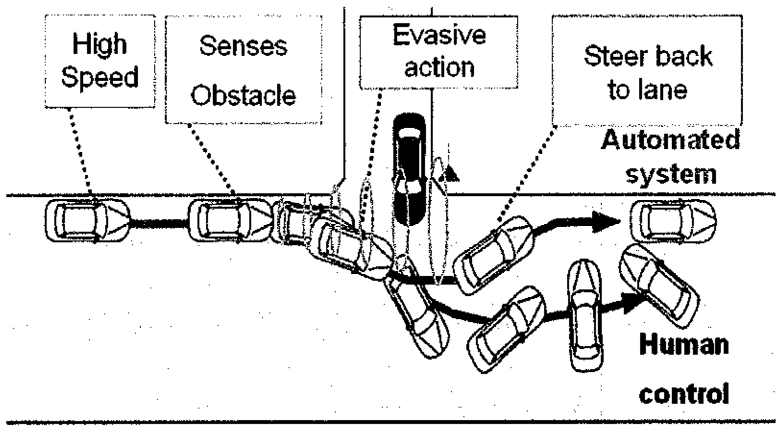


Figure 3.6: Vehicle avoids another vehicle or object appearing unexpectedly in the vehicle's path while traveling on a prescribed path/lane

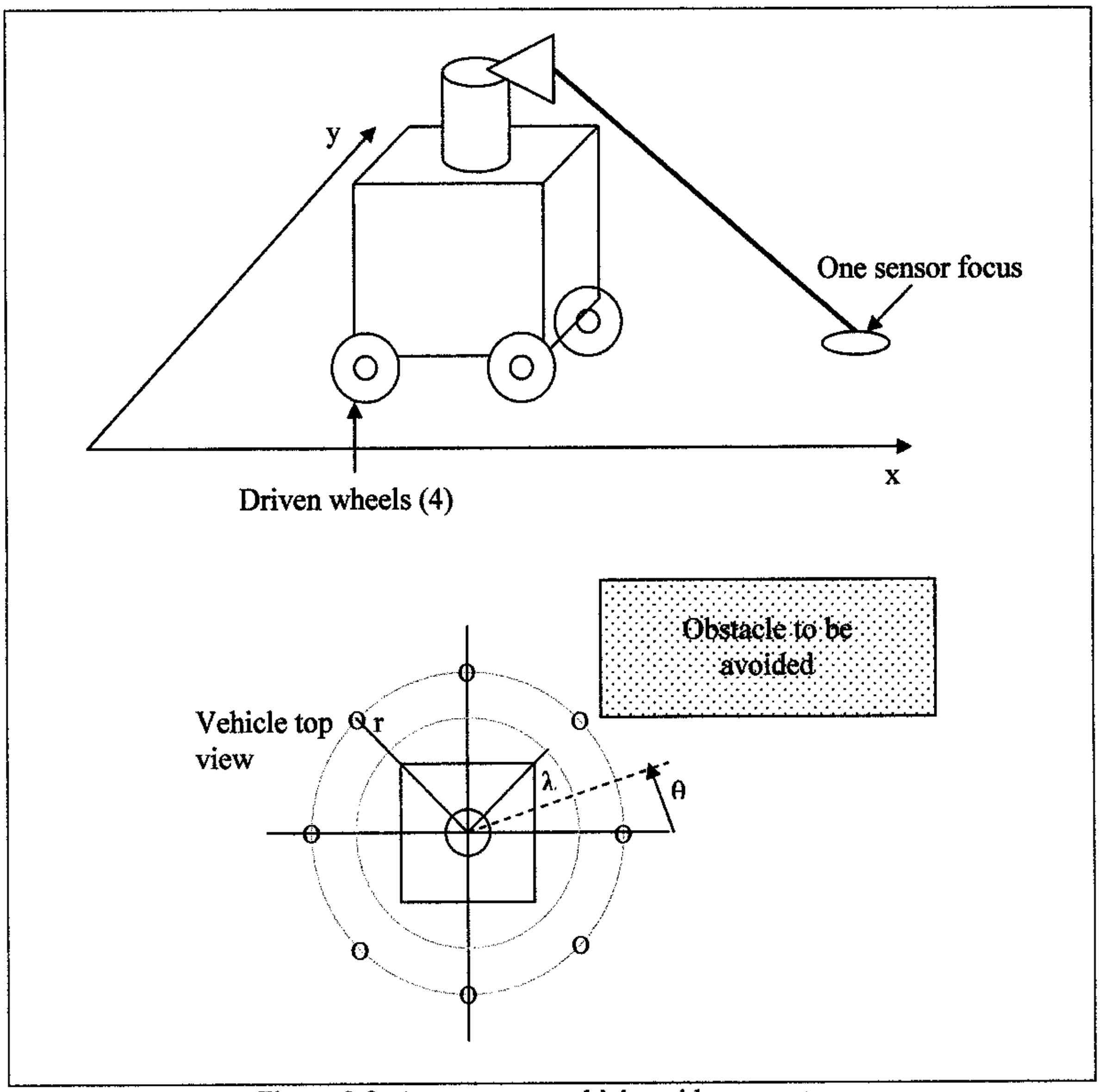


Figure 3.8: Autonomous vehicle guidance system

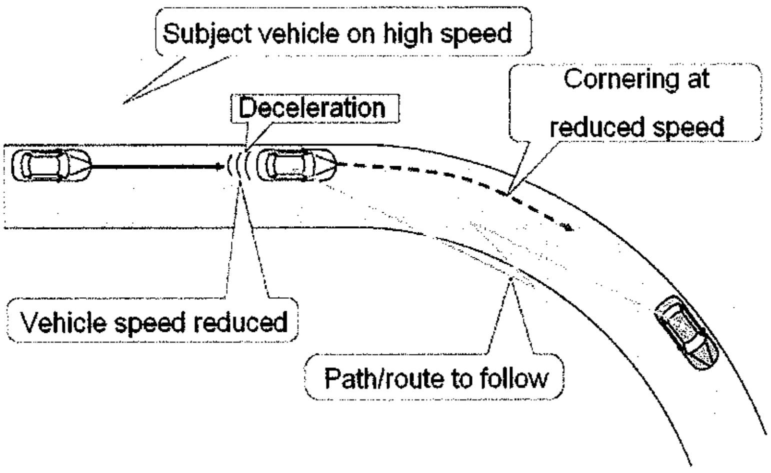


Figure 3.13: Vehicle follows a prescribed path or keeps in a designated lane

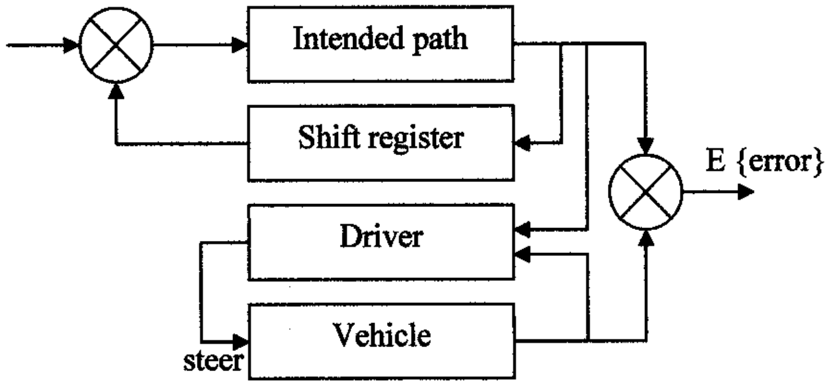


Figure 3.15: Car/road system structure (redrawn from R.S. Sharp and V. Valtetsiotis, 2001)

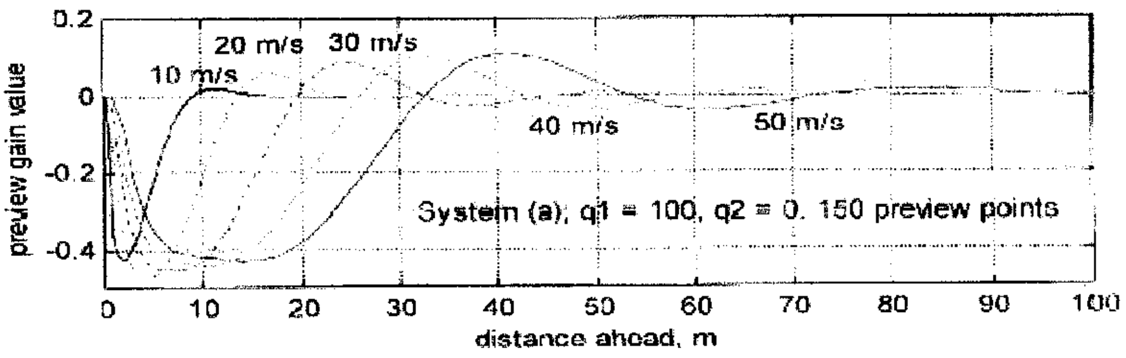


Figure 3.16: Optimal preview gains for path following for five different speeds (taken from N. H. H. Mohamad Hanif, 2004)

Figure 4.7: Obstacle course (straight path) at 120 km/h with 120 preview points

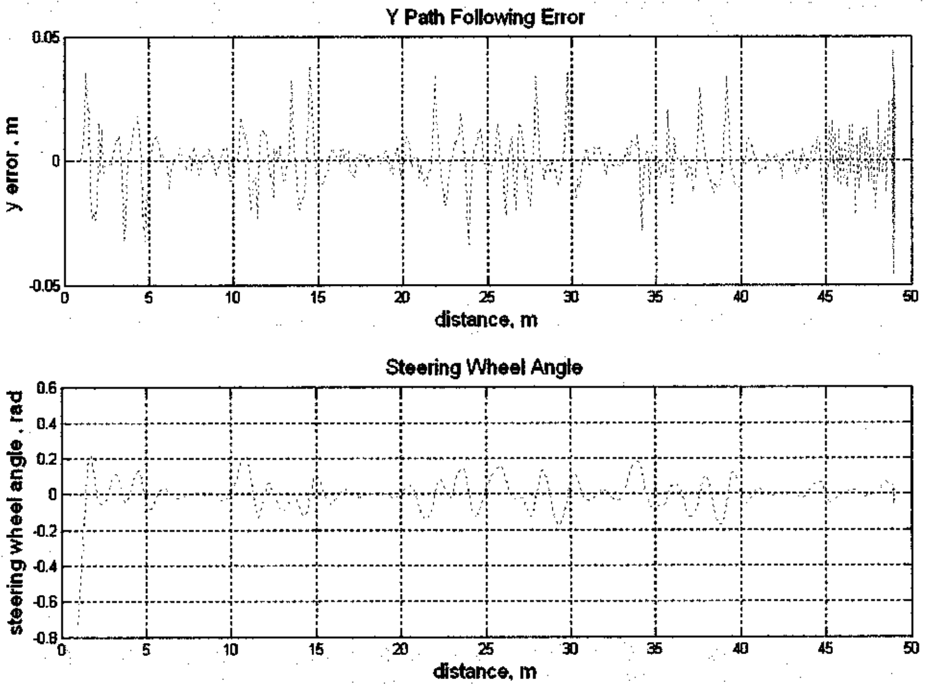


Figure 4.7(i) – (top): Y path following error

Figure 4.7(ii) – (bottom): Steering wheel angle

Figure 4.8: Obstacle course (straight path) at 130 km/h with 120 preview points

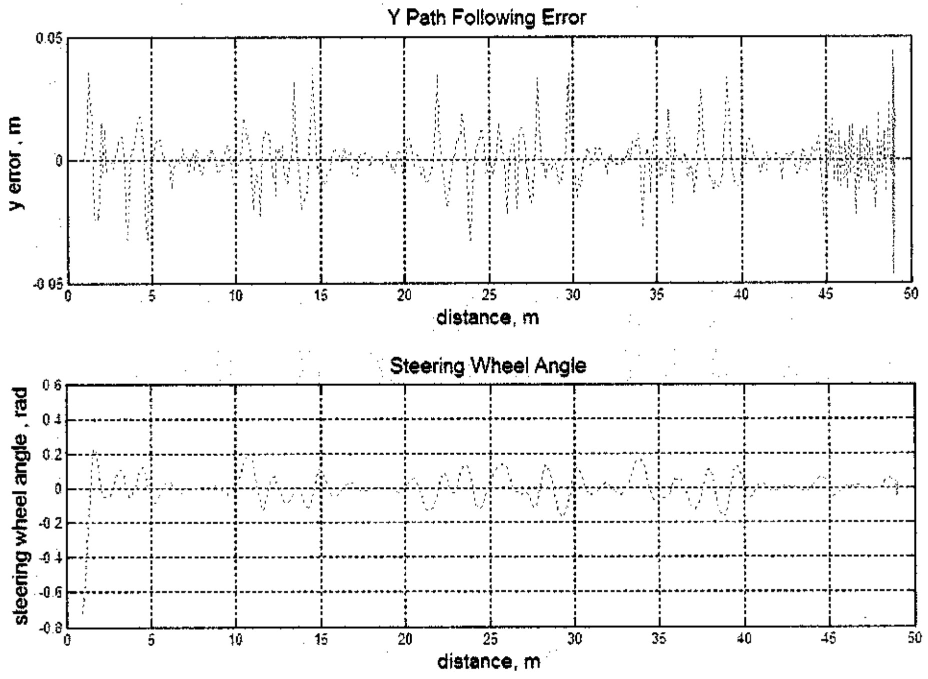
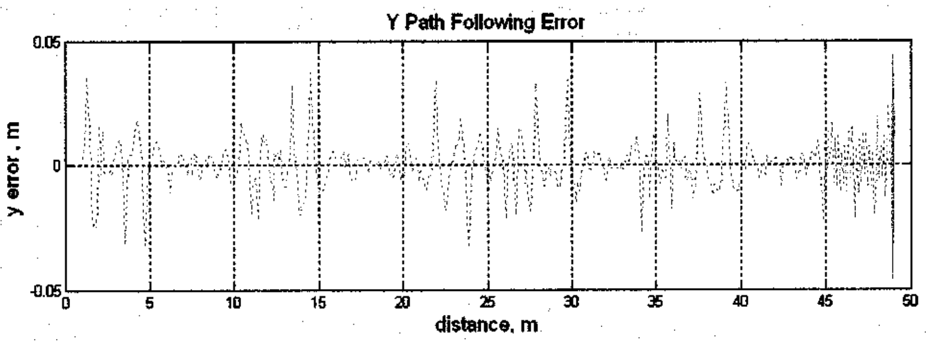


Figure 4.8(i) – (top): Y path following error

Figure 4.8(ii) – (bottom): Steering wheel angle

Figure 4.9: Obstacle course (straight path) at 140 km/h with 120 preview points



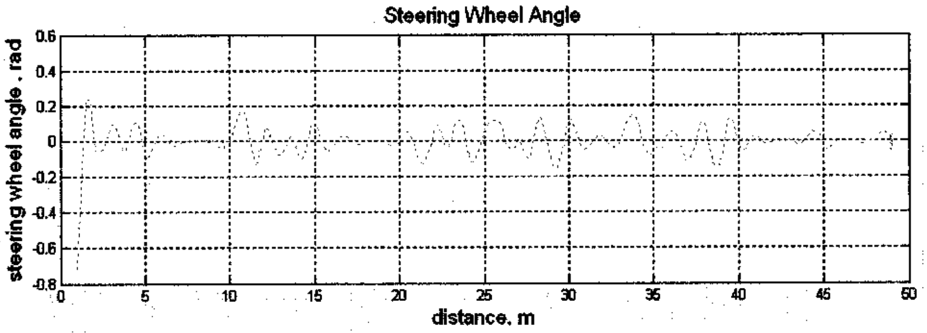
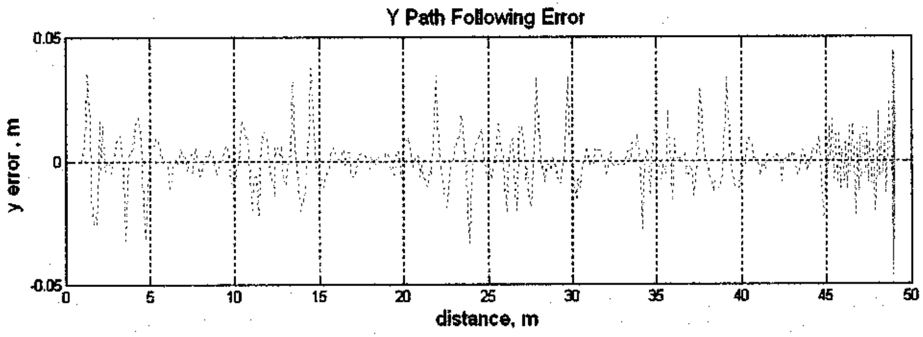


Figure 4.9(i) – (top): Y path following error

Figure 4.9(ii) – (bottom): Steering wheel angle

Figure 4.10: Obstacle course (straight path) at 150 km/h with 120 preview points



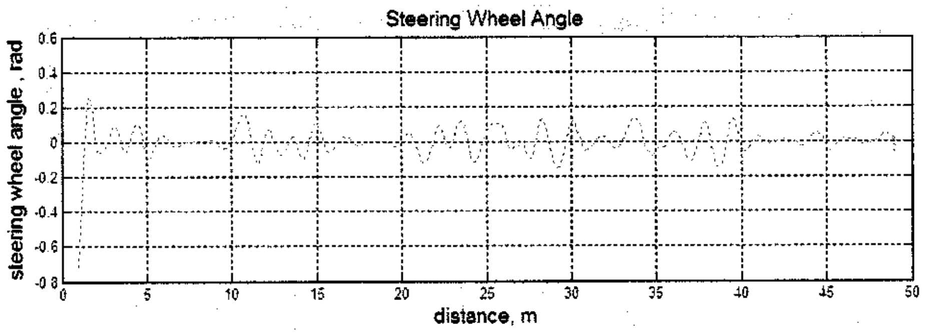


Figure 4.10(i) – (top): Y path following error

Figure 4.10(ii) – (bottom): Steering wheel angle

Figure 4.11: Obstacle course (straight path) at 160 km/h with 120 preview points

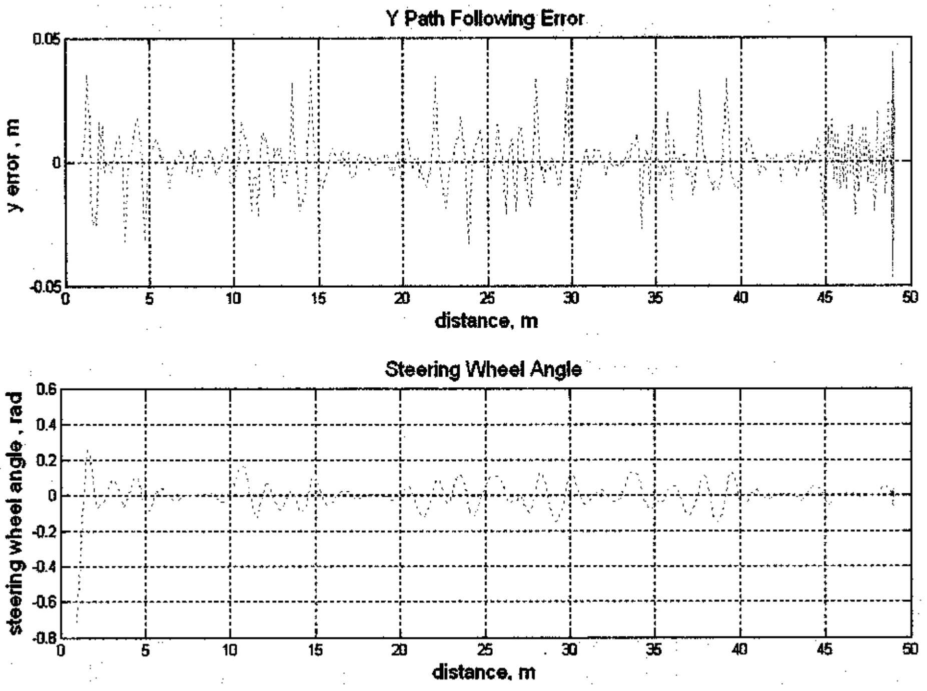


Figure 4.11(i) – (top): Y path following error

Figure 4.11(ii) – (bottom): Steering wheel angle

Figure 4.12: Obstacle course (straight path) at 170 km/h with 120 preview points

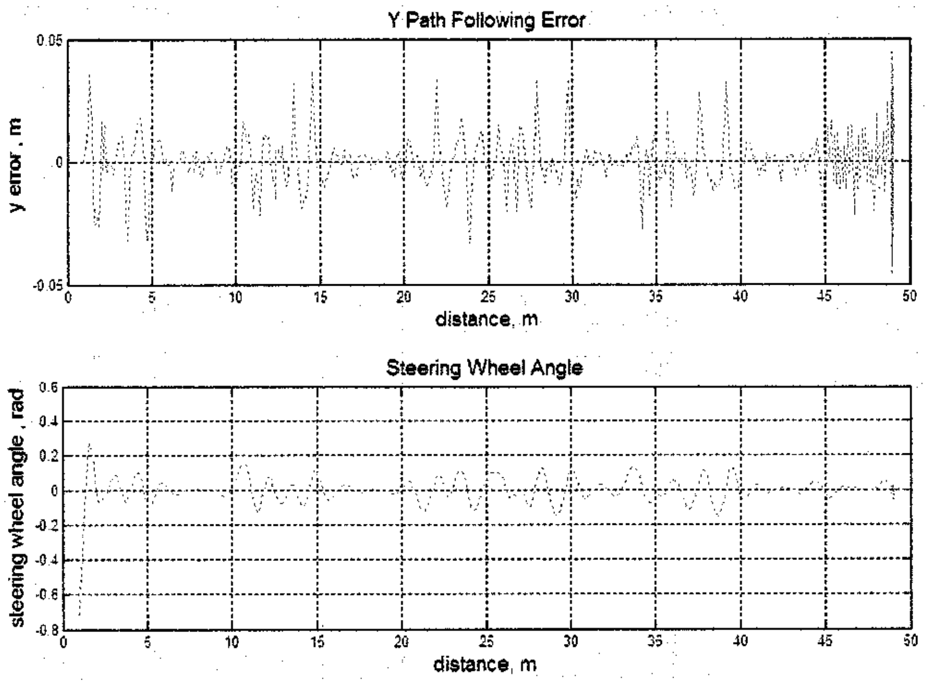


Figure 4.12(i) – (top): Y path following error

Figure 4.12(ii) – (bottom): Steering wheel angle

Figure 4.13: Obstacle course (straight path) at 180 km/h with 120 preview points

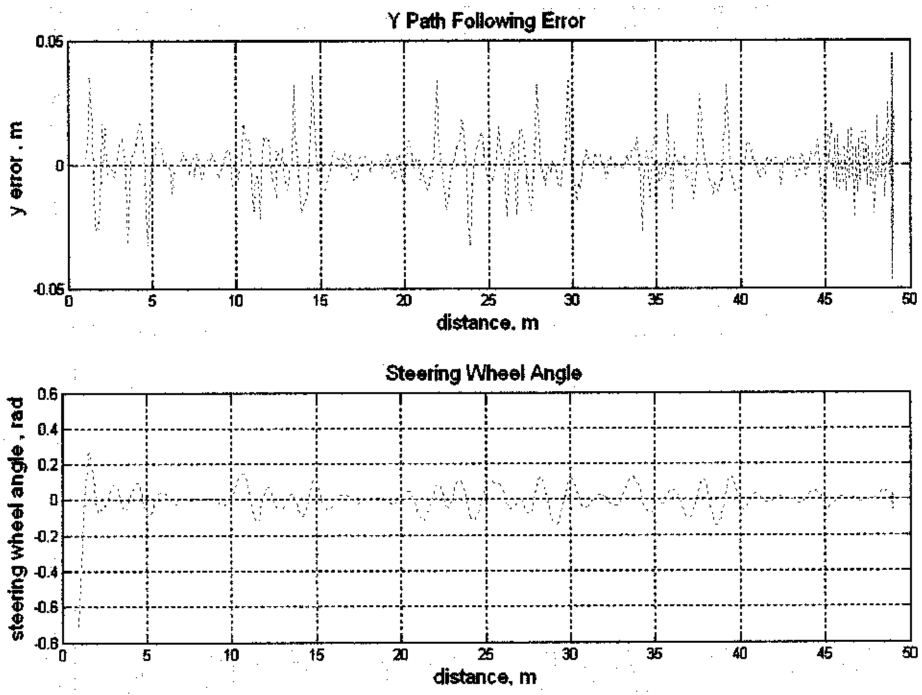


Figure 4.13(i) – (top): Y path following error

Figure 4.13(ii) – (bottom): Steering wheel angle

Figure 4.14: Obstacle course (straight path) at 190 km/h with 120 preview points

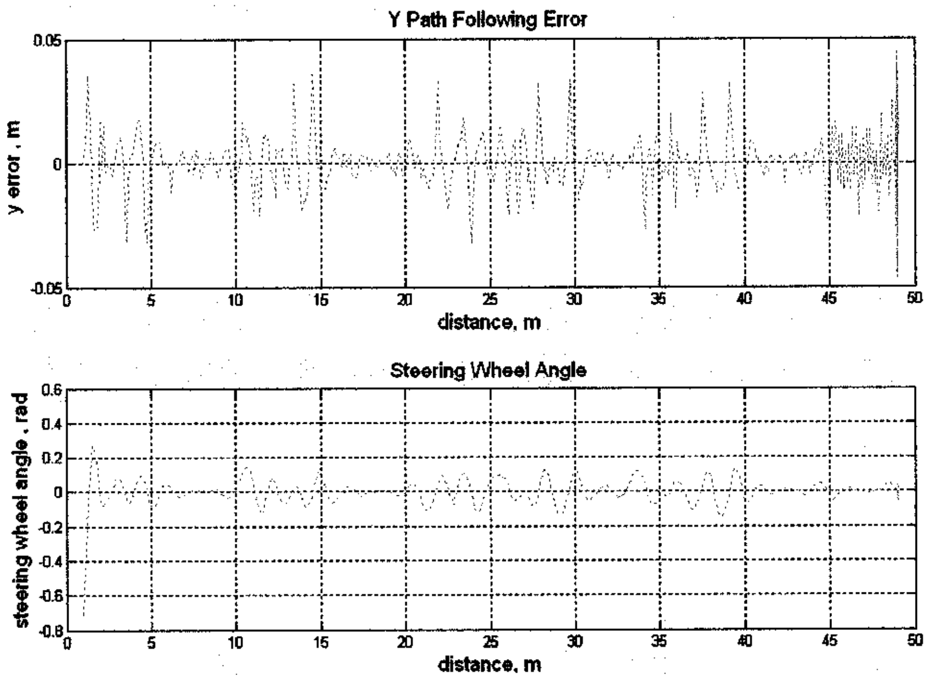


Figure 4.14(i) – (top): Y path following error

Figure 4.14(ii) – (bottom): Steering wheel angle

APPENDIX B

MATLAB Codes

Graphical User Interface (GUI)

```
function varargout = myGUI2(varargin)
% MYGUI2 M-file for myGUI2.fig

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mygui2.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TAN ZHANG YAW
% Electrical & Electronics Department
% Final Year Project
% Universiti Teknologi PETRONAS (UTP)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Neural Network based Controller for High Speed Vehicle following Predetermined Paths Graphical User Interface (GUI)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Last Modified by GUIDE v2.5 14-Oct-2006 15:21:59

%=========================
% INITIALISATION CODES
%=========================

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
   'gui_Singleton',  gui_Singleton, ...
   'gui_OpeningFcn', @myGUI2_OpeningFcn, ...
   'gui_OutputFcn',  @myGUI2_OutputFcn, ...
   'gui_LayoutFcn',  [] , ...
   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

function myGUI2_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;
guidata(hObject, handles);

thisPic = imread('neuralnet.bmp');

axes(handles.myPicture);
imshow(thisPic,'notruesize');
```

```
thisPic2 = imread('cruise.jpg');

axes(handles.myPicture2);
imshow(thisPic2,'notruesize');

function varargout = myGUI2_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

%===========================
% EDIT BOXES INITIAL CODES
%===========================

function ebInitialX_Callback(hObject, eventdata, handles)
function ebInitialX_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebInitialY_Callback(hObject, eventdata, handles)
function ebInitialY_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebGoalX_Callback(hObject, eventdata, handles)
function ebGoalX_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebGoalY_Callback(hObject, eventdata, handles)
function ebGoalY_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle1X_Callback(hObject, eventdata, handles)
function ebObstacle1X_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle1Y_Callback(hObject, eventdata, handles)
function ebObstacle1Y_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle2X_Callback(hObject, eventdata, handles)
function ebObstacle2X_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle2Y_Callback(hObject, eventdata, handles)
function ebObstacle2Y_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function ebObstacle3X_Callback(hObject, eventdata, handles)
function ebObstacle3X_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle3Y_Callback(hObject, eventdata, handles)
function ebObstacle3Y_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle4X_Callback(hObject, eventdata, handles)
function ebObstacle4X_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle4Y_Callback(hObject, eventdata, handles)
function ebObstacle4Y_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle5X_Callback(hObject, eventdata, handles)
function ebObstacle5X_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle5Y_Callback(hObject, eventdata, handles)
function ebObstacle5Y_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle6X_Callback(hObject, eventdata, handles)
function ebObstacle6X_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacle6Y_Callback(hObject, eventdata, handles)
function ebObstacle6Y_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebSpeed_Callback(hObject, eventdata, handles)
function ebSpeed_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebPreview_Callback(hObject, eventdata, handles)
function ebPreview_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function ddPathType_Callback(hObject, eventdata, handles)
function ddPathType_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebObstacleAmount_Callback(hObject, eventdata, handles)
function ebObstacleAmount_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function ebLaneChanges_Callback(hObject, eventdata, handles)
function ebLaneChanges_CreateFcn(hObject, eventdata, handles)
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

%==============================================
% RANDOMISE OBSTACLES CODES
%==============================================

function btnRandomObstacles_Callback(hObject, eventdata, handles)

numberOfObstacles = str2double(get(handles.ebObstacleAmount,'String'));

if numberOfObstacles >= 1
   randomNumber = round(50*rand(1));
   set(handles.ebObstacle1X,'String',randomNumber);
   randomNumber = round(30*rand(1));
   set(handles.ebObstacle1Y,'String',randomNumber);
else
   set(handles.ebObstacle1X,'String','0');
   set(handles.ebObstacle1Y,'String','0');
end

if numberOfObstacles >= 2
   randomNumber = round(50*rand(1));
   set(handles.ebObstacle2X,'String',randomNumber);
   randomNumber = round(30*rand(1));
   set(handles.ebObstacle2Y,'String',randomNumber);
else
   set(handles.ebObstacle2X,'String','0');
   set(handles.ebObstacle2Y,'String','0');
end

if numberOfObstacles >= 3
   randomNumber = round(50*rand(1));
   set(handles.ebObstacle3X,'String',randomNumber);
   randomNumber = round(30*rand(1));
   set(handles.ebObstacle3Y,'String',randomNumber);
else
   set(handles.ebObstacle3X,'String','0');
   set(handles.ebObstacle3Y,'String','0');
end

if numberOfObstacles >= 4
   randomNumber = round(50*rand(1));
   set(handles.ebObstacle4X,'String',randomNumber);
   randomNumber = round(30*rand(1));
   set(handles.ebObstacle4Y,'String',randomNumber);
else
   set(handles.ebObstacle4X,'String','0');
   set(handles.ebObstacle4Y,'String','0');
end
```

```
if numberOfObstacles >= 5
   randomNumber = round(50*rand(1));
   set(handles.ebObstacle5X,'String',randomNumber);
   randomNumber = round(30*rand(1));
   set(handles.ebObstacle5Y,'String',randomNumber);
else
   set(handles.ebObstacle5X,'String','0');
   set(handles.ebObstacle5Y,'String','0');
end

if numberOfObstacles >= 6
   randomNumber = round(50*rand(1));
   set(handles.ebObstacle6X,'String',randomNumber);
   randomNumber = round(30*rand(1));
   set(handles.ebObstacle6Y,'String',randomNumber);
else
   set(handles.ebObstacle6X,'String','0');
   set(handles.ebObstacle6Y,'String','0');
end
set(handles.txtCurrentLane,'String','1');

%============
% RESET CODES
%============

function btnResetAll_Callback(hObject, eventdata, handles)

set(handles.ebInitialX,'String','0');
set(handles.ebInitialY,'String','0');
set(handles.ebGoalX,'String','50');
set(handles.ebGoalY,'String','30');

set(handles.ebObstacle1X,'String','0');
set(handles.ebObstacle1Y,'String','0');
set(handles.ebObstacle2X,'String','0');
set(handles.ebObstacle2Y,'String','0');
set(handles.ebObstacle3X,'String','0');
set(handles.ebObstacle3Y,'String','0');
set(handles.ebObstacle4X,'String','0');
set(handles.ebObstacle4Y,'String','0');
set(handles.ebObstacle5X,'String','0');
set(handles.ebObstacle5Y,'String','0');
set(handles.ebObstacle6X,'String','0');
set(handles.ebObstacle6Y,'String','0');

set(handles.ebLaneChanges,'String','0');
set(handles.ebObstacleAmount,'String','6');
set(handles.txtFinalError,'String','Average Error = 0'); %reset Average Error to 0

set(handles.ebSpeed,'String','90');
set(handles.ebPreview,'String','20');

axes(handles.boxObstaclePreview);
cla;
axes(handles.boxShowPath);
cla;
axes(handles.boxPath);
cla;
axes(handles.boxError);
cla;

set(handles.txtCurrentLane,'String','1');

%====================
% INFORMATION CODES 1
%====================

function btnPreviewObstacles_Callback(hObject, eventdata, handles)

% Grab initial and end vehicle coordinates
```

```
initialX = str2double(get(handles.ebInitialX,'String'));
initialY = str2double(get(handles.ebInitialY,'String'));
goalX = str2double(get(handles.ebGoalX,'String'));
goalY = str2double(get(handles.ebGoalY,'String'));

% Get the projected line
if initialX == goalX
  xValues = (initialX-0.001):0.001/100:goalX;
else
  xValues = initialX:(goalX-initialX)/100:goalX;
end
if initialY == goalY
  yValues = (initialY-0.001):0.001/100:goalY;
else
  yValues = initialY:(goalY-initialY)/100:goalY;
end

% Grab obstacle coordinates
obs1X = str2double(get(handles.ebObstacle1X,'String'));
obs1Y = str2double(get(handles.ebObstacle1Y,'String'));
obs2X = str2double(get(handles.ebObstacle2X,'String'));
obs2Y = str2double(get(handles.ebObstacle2Y,'String'));
obs3X = str2double(get(handles.ebObstacle3X,'String'));
obs3Y = str2double(get(handles.ebObstacle3Y,'String'));
obs4X = str2double(get(handles.ebObstacle4X,'String'));
obs4Y = str2double(get(handles.ebObstacle4Y,'String'));
obs5X = str2double(get(handles.ebObstacle5X,'String'));
obs5Y = str2double(get(handles.ebObstacle5Y,'String'));
obs6X = str2double(get(handles.ebObstacle6X,'String'));
obs6Y = str2double(get(handles.ebObstacle6Y,'String'));

% Plot initial and final positions
axes(handles.boxObstaclePreview);
plot(xValues,yValues);
axis([0 50 0 30]);
title('Initial Vehicle Position, Goal Position & Obstacles');

% Plot obstacle positions (sets obstaclefunction)
hold on
if obs1X > 0
  if obs1Y > 0
    plot(obs1X,obs1Y,'ro')
  end
end
hold off
hold on
if obs2X > 0
  if obs2Y > 0
    plot(obs2X,obs2Y,'ro')
  end
end
hold off
hold on
if obs3X > 0
  if obs3Y > 0
    plot(obs3X,obs3Y,'ro')
  end
end
hold off
hold on
if obs4X > 0
  if obs4Y > 0
    plot(obs4X,obs4Y,'ro')
  end
end
hold off
hold on
if obs5X > 0
  if obs5Y > 0
    plot(obs5X,obs5Y,'ro')
```

```
   end
end
hold off
hold on
if obs6X > 0
   if obs6Y > 0
      plot(obs6X,obs6Y,'ro')
   end
end
hold off

%=====================
% LANE CHANGE CODES
%=====================

function btnStart_Callback(hObject, eventdata, handles)

% --- Setup all the initial variables ---
numberOfLanes = str2double(get(handles.ebLaneChanges,'String')) + 1;
if numberOfLanes > 3
   numberOfLanes = 3;
end
currentLane = str2double(get(handles.txtCurrentLane,'String'));

% Grab initial and end vehicle coordinates
initialX = str2double(get(handles.ebInitialX,'String'));
initialY = str2double(get(handles.ebInitialY,'String'));
goalX = str2double(get(handles.ebGoalX,'String'));
goalY = str2double(get(handles.ebGoalY,'String'));

% Get the projected line
if initialX == goalX
   xValues = (initialX-0.001):0.001/100:goalX;
else
   xValues = initialX:(goalX-initialX)/100:goalX;
end
if initialY == goalY
   yValues = (initialY-0.001):0.001/100:goalY;
else
   yValues = initialY:(goalY-initialY)/100:goalY;
end

val = get(handles.ddPathType,'Value');
str = get(handles.ddPathType, 'String');
thisMethod = str{val};

% Grab obstacle coordinates
obs1X = str2double(get(handles.ebObstacle1X,'String'));
obs1Y = str2double(get(handles.ebObstacle1Y,'String'));
obs2X = str2double(get(handles.ebObstacle2X,'String'));
obs2Y = str2double(get(handles.ebObstacle2Y,'String'));
obs3X = str2double(get(handles.ebObstacle3X,'String'));
obs3Y = str2double(get(handles.ebObstacle3Y,'String'));
obs4X = str2double(get(handles.ebObstacle4X,'String'));
obs4Y = str2double(get(handles.ebObstacle4Y,'String'));
obs5X = str2double(get(handles.ebObstacle5X,'String'));
obs5Y = str2double(get(handles.ebObstacle5Y,'String'));
obs6X = str2double(get(handles.ebObstacle6X,'String'));
obs6Y = str2double(get(handles.ebObstacle6Y,'String'));

speed = str2double(get(handles.ebSpeed,'String'));
speed = speed*1000/3600;     % convert km/h to m/s
previewPoints = str2double(get(handles.ebPreview,'String'));

%=====================
% PREVIEW CODES
%=====================

% Plot initial and final positions
axes(handles.boxObstaclePreview);
```

```
plot(xValues,yValues);
axis([0 50 0 30]);
title('Initial Vehicle Position, Goal Position & Obstacles');

% Plot obstacle positions (sets obstaclefunction)
hold on
if obs1X > 0 && obs1Y > 0
   plot(obs1X,obs1Y,'ro')
end
hold off
hold on
if obs2X > 0 && obs2Y > 0
   plot(obs2X,obs2Y,'ro')
end
hold off
hold on
if obs3X > 0 && obs3Y > 0
   plot(obs3X,obs3Y,'ro')
end
hold off
hold on
if obs4X > 0 && obs4Y > 0
   plot(obs4X,obs4Y,'ro')
end
hold off
hold on
if obs5X > 0 && obs5Y > 0
   plot(obs5X,obs5Y,'ro')
end
hold off
hold on
if obs6X > 0 && obs6Y > 0
   plot(obs6X,obs6Y,'ro')
end
hold off

obs(1) = obs1X; obs(2) = obs1Y;
obs(3) = obs2X; obs(4) = obs2Y;
obs(5) = obs3X; obs(6) = obs3Y;
obs(7) = obs4X; obs(8) = obs4Y;
obs(9) = obs5X; obs(10) = obs5Y;
obs(11) = obs6X; obs(12) = obs6Y;

%===================================
% OBSTACLE AVOIDANCE CODES
%===================================

% Set edges of region want to search in
xmin=[0; 0];
xmax=[50;30];
Nsteps=750;      % Maximum number of steps to produce
lambda=0.1;      % Step size to take in chosen direction at each move
Ns=16;           % Number of points on circular pattern to sense
r=1;             % Sensing radius
xs=0*ones(2,Ns);  % Initialize xs
Jo(:,1)=0*ones(Ns,1);
Jg(:,1)=0*ones(Ns,1);
J(:,1)=0*ones(Ns,1);
theta(:,1)=0*ones(Ns,1);
for m=2:Ns       % Compute the angles to be used around the circle
   theta(m,1)=theta(m-1,1)+(pi/180)*(360/Ns);
end

% Goal position of vehicle
xgoal=[goalX; goalY];
% Initial vehicle position
x=[initialX; initialY];

% Weighting parameters for planning (sets priority for being aggresive
% in the direction of the goal vs. avoiding obstacles
```

```
w1=1;
w2=1.0000e-04;

% Allocate memory
x(:,2:Nsteps)=0*ones(2,Nsteps-1);

xx=0:50/100:50;
yy=0:30/100:30;

for jj=1:length(xx)
   for ii=1:length(yy)
      zz(ii,jj)=gui_obsfunction([xx(jj);yy(ii)],w1,obs);
   end
end
for jj=1:length(xx)
   for ii=1:length(yy)
      zzz(ii,jj)=goalfunction([xx(jj);yy(ii)],xgoal,w2);
   end
end

for k=1:Nsteps

   % Use projection to keep in boundaries (like hitting a wall and staying at it)

   x(:,k)=min(x(:,k),xmax);
   x(:,k)=max(x(:,k),xmin);

   % Sense points on circular pattern

   for m=1:Ns
      xs(:,m)=[x(1,k)+r*cos(theta(m,1)); x(2,k)+r*sin(theta(m,1))]; % Point on circular pattern
      Jo(m,1)=gui_obsfunction(xs(:,m),w1,obs); % Compute the obstace function (what is
      % sensed at each sensed point
      Jg(m,1)=goalfunction(xs(:,m),xgoal,w2); % Compute how well each point
      % moves toward the goal
      J(m,1)=Jo(m,1)+Jg(m,1); % Compute function for opt. in planning
   end

   % Next pick the best direction

   [val,bestone]=min(J);

   % Then, update the vehicle position (pick best direction and move step of lambda that way)

   x(:,k+1)=[x(1,k)+lambda*cos(theta(bestone,1)); x(2,k)+lambda*sin(theta(bestone,1))];

   % But the vehicle is in a real environment so when it tries to move to that point it
   % only gets to near that point. To simulate this we perterb the final position.

   Deltalambda=0.1*lambda*(2*rand-1); % Set the length perturbation to be up to 10% of the step size
   Deltatheta=2*pi*(2*rand-1); % Set to be 360deg variation from chosen direction
   x(:,k+1)=[x(1,k+1)+Deltalambda*cos(theta(bestone,1)+Deltatheta); ...
      x(2,k+1)+Deltalambda*sin(theta(bestone,1)+Deltatheta)];

end % End main loop...

%==================
% DISPLAY PATH CODES
%==================

axes(handles.boxShowPath);
contour(xx,yy,zz,25);
colormap(jet);
title('Vehicle Path');
if currentLane > 1
   load PathOne
   hold on
   plot(first_x(1,:),first_x(2,:),'r-');
   hold off
end
```

```
if currentLane > 2
   load PathTwo
   hold on
   plot(second_x(1,:),second_x(2,:),'r-');
   hold off
end
hold on
plot(x(1,:),x(2,:),'r-');
plot(initialX,initialY,'s',goalX,goalY,'x');
hold off

if currentLane < numberOfLanes
   if currentLane == 1
      first_x = x;
      save PathOne first_x
      set(handles.ebInitialX,'String',get(handles.ebGoalX,'String'));
      set(handles.ebInitialY,'String',get(handles.ebGoalY,'String'));
   end
   if currentLane == 2
      second_x = x;
      save PathTwo second_x
      set(handles.ebInitialX,'String',get(handles.ebGoalX,'String'));
      set(handles.ebInitialY,'String',get(handles.ebGoalY,'String'));
   end
   currentLane = currentLane + 1;
   set(handles.txtCurrentLane,'String',num2str(currentLane));
end
save PathThis x


%%%%%%%%%% End of Program %%%%%%%%%%%


function btnError_Callback(hObject, eventdata, handles)

%====================================
% INFORMATION CODES 2
%====================================

%axes(handles.boxPath);
%title('Follow Path');
%plot(x(1,:),x(2,:),'r-');

speed = str2double(get(handles.ebSpeed,'String'));
speed = speed*1000/3600;      % convert km/h to m/s
previewPoints = str2double(get(handles.ebPreview,'String'));
currentLane = str2double(get(handles.txtCurrentLane,'String'));
val = get(handles.ddPathType,'Value');
str = get(handles.ddPathType, 'String');
thisMethod = str{val};

load PathThis
[myWidth,myLength] = size(x);
if currentLane > 1
   load PathOne
   x(1,myLength+1:myLength*2) = x(1,1:myLength);
   x(1,1:myLength) = first_x(1,1:myLength);
   x(2,myLength+1:myLength*2) = x(2,1:myLength);
   x(2,1:myLength) = first_x(2,1:myLength);
end
if currentLane > 2
   load PathTwo
   x(1,myLength*2+1:myLength*3) = x(1,myLength+1:myLength*2);
   x(1,myLength+1:myLength*2) = second_x(1,1:myLength);
   x(2,myLength*2+1:myLength*3) = x(2,myLength+1:myLength*2);
   x(2,myLength+1:myLength*2) = second_x(2,1:myLength);
end

axes(handles.boxPath);
title('Follow Path');
```

```
plot(x(1,:),x(2,:),'r-');

% forward speed
u=speed;
% sampling period T
T=0.05;
% number of preview points
n=previewPoints;
%car parameters definition
Cf=120000;
Cr=80000;
a=0.92;
b=1.38;
M=1200;
G=17;
Iz=1500;

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

% road shift operators
D=[zeros(n,1) eye(n);zeros(1,n+1)];
E=[zeros(n,1); 1];

%D=[0 1 0 0 0 0 0 0 0 0 ;
% 0 0 1 0 0 0 0 0 0 0 ;
% 0 0 0 1 0 0 0 0 0 0 ;
% 0 0 0 0 1 0 0 0 0 0 ;
% 0 0 0 0 0 1 0 0 0 0 ;
% 0 0 0 0 0 0 1 0 0 0 ;
% 0 0 0 0 0 0 0 1 0 0 ;
% 0 0 0 0 0 0 0 0 1 0 ;
% 0 0 0 0 0 0 0 0 0 1 ;
% 0 0 0 0 0 0 0 0 0 0 ];
%E=[0;0;0;0;0;0;0;0;0;1];

%%%%%%%%%% Linear Car Model %%%%%%%%%%%

Linear_car_model

%%%%%%%%%% Linear Control Gain Calculation %%%%%%%%%%%

% 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)
% We keep the same priorites.
R1cost=R1;
R2cost=R2;
tic    %starts the stopwatch timer

%%%%%%%%%%% Path Information %%%%%%%%%%%

for epoch = 1:5

   switch(thisMethod)
      case 'Sinus Shape'
         % sinus shape
         xref=[0:u*T:900];
         yref=50*sin(xref/100);

      case 'Lane Change'
         %lane change
         xr1=[0:u*T:50-u*T];
      xr2=[50:u*T:50+60];
      xr3=[110+u*T:u*T:300];
      xref=[ xr1 xr2 xr3];
      yr1=0*xr1;
      yr2=2-2*cos((pi/60)*xr2-50*pi/60);
      yr3=4*ones(size(xr3));
      yref=[yr1 yr2 yr3];

   case 'Sudden Change of Direction'
      %sudden change of direction
      xr1=[0:u*T:60-u*T];
      xr2=[60:u*T:200];
      xref=[xr1 xr2];
      yr1 = 0*xr1;
      SIZE_OF_xr2=size(xr2);
      yr2 = 0.5*0.25*[1:SIZE_OF_xr2(1,2)]*0.5359';
      yref = [yr1 yr2];

   case 'Smooth Random Path'
      %smooth random path
      K=900/(u*T)+1;
      xref=[0:u*T:900];
      [Bfilter, Afilter] = butter(5, 0.007);
      roadn=10*rand(K,1);
      roadn = 40*(roadn-5);
      yref = filter(Bfilter, Afilter, roadn)';

   case 'Obstacle Avoidance'
      %red-line path
      xref = x(1,:);
      yref(1,:) = x(2,:);
end

%if epoch ==1
%circuits_2
%else
%circuit_iterations
%end

[K,nb] = size (yref'); %array size of yref

%%%%%%%%%%%% State Definition & Initialisation %%%%%%%%%%%%

% At each time 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 ]

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 matrix
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+5 (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 the local x and y axes of the car.

   % definition of the state 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 is changed
   ZinitB(2) = ZinitB(2)-u*sin((phiB(k)-phiB(k-1)));
   %ZinitC(2) = ZinitC(2)-u*sin((phiC(k)-phiC(k-1)));  %the local y-axis is changed

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 remainning 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 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
if dJdw/prevdJdw < 1.000  %cost ratio
   gama=1.05*gama;
end
if dJdw/prevdJdw > 1.005
```

```
            gama=0.7*gama;
        end
        % difference calculation Polak Ribiere or Gradient method
        % prevdeltaw=deltaw;
        % if k<3
        deltaw=-gama*dJdw; %value for deltaw
        gama_next(k+1)=gama;

        % else
        % deltaw=-gama*(dJdw - 1/norm(prevdJdw) *...
        % (dJdw'*(dJdw-prevdJdw)*prevdeltaw')');
        % end

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

        %%%%%%%%%%%% End of Weighting 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 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Plottings2inoneshot

road_angle_estimation=zeros([1:K-n+1],1);
for j=1:K-n
    road_angle_estimation(j)=(yref(j+1)-yref(j))/(u*T);
end
yaw_errorA=phiA([1:K-n])-road_angle_estimation([1:K-n]);

axes(handles.boxPath);
%path following
plot(xref([1:K-n]),yref(1,[1:K-n]),xref([1:K-n]),global_positionA(1:K-n),'g:');
%plot(xglobal([1:K-n+1]),yref(1,[1:K-n+1]),xglobal([1:K-n+1]),yglobal(1:K-n+1,1),'g');grid;
xlabel('distance, m');
```

```
ylabel('y coordinate , m');
title('Path Following');
switch(thisMethod)
   case 'Sinus Shape'
      if i==2
         AXIS([-10 1000 -100 100])
      end

   case 'Lane Change'
      if i==2
         AXIS([-10 1000 -100 100])
      end

   case 'Sudden Change of Direction'
      if i==2
         AXIS([-10 1000 -100 100])
      end

   case 'Smooth Random Path'
      if i==2
         AXIS([-10 1000 -100 100])
      end

   case 'Obstacle Avoidance'
      axis([0 50 0 30]);
end

axes(handles.boxError);
%y difference
plot(xref([1:K-n]),yref([1:K-n])-global_positionA([1:K-n]),':');
xlabel('distance, m');
ylabel('y error , m ');
title('Y Path Following Error');

myCount = 1;
averageError = 0;
totalError = 0;
thisError = 0;
while myCount < (K-n)
   thisError = yref(myCount) - global_positionA(myCount);
   if thisError < 0
      thisError = -thisError;
   end
   totalError = totalError + thisError;
   myCount = myCount + 1;
end
averageError = totalError/myCount;

set(handles.txtFinalError,'String',['Average Error = ' num2str(averageError)]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Obstacle Collision Avoidance

Main Program

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% vehic_guide.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Obstacle Collision Avoidance
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear           % Initialize memory

xmin=[0; 0];      % Set edges of region want to search in
xmax=[50;30];

Nsteps=750;        % Maximum number of steps to produce

% Next set the parameters of the vehicle:

lambda=0.1; % Step size to take in chosen direction at each move
Ns=16;      % Number of points on circular pattern to sense
r=1;        % Sensing radius
xs=0*ones(2,Ns); % Initialize
Jo(:,1)=0*ones(Ns,1);
Jg(:,1)=0*ones(Ns,1);
J(:,1)=0*ones(Ns,1);
theta(:,1)=0*ones(Ns,1);
for m=2:Ns  % Compute the angles to be used around the circle
          theta(m,1)=theta(m-1,1)+(pi/180)*(360/Ns);
end

% Goal position of vehicle
xgoal=[49; 15];

% Initial vehicle position
x=[1; 15];

% Weighting parameters for planning (sets priority for being aggresive
% in the direction of the goal vs. avoiding obstacles
w1=1;
w2=1.0000e-04;

% Allocate memory

x(:,2:Nsteps)=0*ones(2,Nsteps-1);

% The obstacles:

figure(1)
clf
% Plot initial and final positions
plot(1,15,'s',49,15,'x')
axis([0 50 0 30])
hold on
xlabel('x');
ylabel('y');
title('Obstacles (o), initial vehicle (square) and goal (x) positions');
hold on
% Plot obstacle positions (sets obstaclefunction)
plot(5,15,'o',15,18,'o',15,12,'o',25,17,'o',30,13,'o',38,15,'o')
hold off
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the functions:

xx=0:50/100:50;  % For our function the range of values we are considering
yy=0:30/100:30;

% Compute the obstacle and goal functions

for jj=1:length(xx)
        for ii=1:length(yy)
                zz(ii,jj)=obstaclefunction([xx(jj);yy(ii)],w1);
        end
end
for jj=1:length(xx)
        for ii=1:length(yy)
                zzz(ii,jj)=goalfunction([xx(jj);yy(ii)],xgoal,w2);
        end
end

figure(2)
clf
surf(xx,yy,zz);
%colormap(jet)
% Use next line for generating plots to put in black and white documents.
colormap(white);
xlabel('x');
ylabel('y');
zlabel('w_1J_o');
title('Function w_1J_o showing (scaled) obstacle function values');


figure(3)
clf
contour(xx,yy,zz,25)
colormap(jet)
% Use next line for generating plots to put in black and white documents.
%colormap(white);
xlabel('x');
ylabel('y');
title('Contour map of w_1J_o and initial (square) and goal (x) positions');
hold on
% Plot initial and final positions
plot(1,15,'s',49,15,'x')
hold off

figure(4)
clf
surf(xx,yy,zzz);
view(82,26);
%colormap(jet)
% Use next line for generating plots to put in black and white documents.
colormap(white);
xlabel('x');
ylabel('y');
zlabel('w_2J_g');
title('Goal function (scaled)');
%rotate3d

figure(5)
clf
contour(xx,yy,zzz,25)
colormap(jet)
% Use next line for generating plots to put in black and white documents.
%colormap(gray);
xlabel('x');
ylabel('y');
title('Contour function of w_2J_g and initial (square) and goal (x) positions');
hold on
% Plot initial and final positions
plot(1,15,'s',49,15,'x')
```

```
hold off


figure(6)
clf
contour(xx,yy,zz+zzz,50)
colormap(jet)
% Use next line for generating plots to put in black and white documents.
%colormap(gray);
xlabel('x');
ylabel('y');
title('J=w_1J_o + w_2J_g and initial (square) and goal (x) positions');
hold on

% Plot initial and final positions
plot(1,15,'s',49,15,'x')

hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Start the simulation loop

for k=1:Nsteps

        % Use projection to keep in boundaries (like hitting a wall and staying at it)

        x(:,k)=min(x(:,k),xmax);
        x(:,k)=max(x(:,k),xmin);

        % Sense points on circular pattern

        for m=1:Ns
                xs(:,m)=[x(1,k)+r*cos(theta(m,1)); x(2,k)+r*sin(theta(m,1))]; % Point on circular pattern
                Jo(m,1)=obstaclefunction(xs(:,m),w1); % Compute the obstace function (what is
                                                                                                   % sensed
at each sensed point
                Jg(m,1)=goalfunction(xs(:,m),xgoal,w2); % Compute how well each point

        % moves toward the goal
                J(m,1)=Jo(m,1)+Jg(m,1); % Compute function for opt. in planning
        end

        % Next pick the best direction

        [val,bestone]=min(J);

        % Then, update the vehicle position (pick best direction and move step of lambda that way)

        x(:,k+1)=[x(1,k)+lambda*cos(theta(bestone,1)); x(2,k)+lambda*sin(theta(bestone,1))];

        % But the vehicle is in a real environment so when it tries to move to that point it
        % only gets to near that point.  To simulate this we perterb the final position.

        Deltalambda=0.1*lambda*(2*rand-1); % Set the length perturbation to be up to 10% of the step size
        Deltatheta=2*pi*(2*rand-1); % Set to be 360deg variation from chosen direction
        x(:,k+1)=[x(1,k+1)+Deltalambda*cos(theta(bestone,1)+Deltatheta); ...
            x(2,k+1)+Deltalambda*sin(theta(bestone,1)+Deltatheta)];

end % End main loop...


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Next, provide some plots of the results of the simulation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t=0:Nsteps;  % For use in plotting

figure(7)
clf
```

```
plot(t,x(1,:),'k-',t,x(2,:),'k--')
ylabel('x, y')
xlabel('Iteration, k')
title('Vehicle trajectory (x solid, y dashed)')


figure(8)
clf
contour(xx,yy,zz,25)
colormap(jet)
% Use next line for generating plots to put in black and white documents.
%colormap(gray);
xlabel('x');
ylabel('y');
title('Vehicle path to avoid obstacles and reach goal');

hold on

plot(x(1,:),x(2,:),'r-')

plot(1,15,'s',49,15,'x')

hold off

save dirResult x

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Obstacle Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% obstaclefunction.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Obstacle Function
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function J=obstaclefunction(x,w1)

% A function to represent sensed obstacles:

        J=...
                w1*max([exp(-0.8*((x(1,1)-5)^2+(x(2,1)-15)^2)),...
                exp(-0.8*((x(1,1)-15)^2+(x(2,1)-18)^2)),...
                exp(-0.8*((x(1,1)-15)^2+(x(2,1)-12)^2)),...
                exp(-0.8*((x(1,1)-25)^2+(x(2,1)-17)^2)),...
                exp(-0.8*((x(1,1)-30)^2+(x(2,1)-13)^2)),...
                exp(-0.8*((x(1,1)-38)^2+(x(2,1)-15)^2))]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Goal Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% goalfunction.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% Goal Function
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Jg=goalfunction(x,xgoal,w2)

% A goal function:

        Jg=w2*(x-xgoal)'*(x-xgoal);
%       Jg=0.01*(x-xgoal)'*(x-xgoal)/(1-0.01*(x-xgoal)'*(x-xgoal));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Path Following/Lane Keeping

Main Program

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LinearNNa.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Neural Control of a Linear Car Model with a Single Processing Element
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


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

% forward speed
u=input('Speed in km/h ? (using 20 par (default))');
u = u*1000/3600;      % convert km/h to m/s
if isempty(u), u=20; disp('Using u=20m/s (default)'), end

% sampling period T
T=0.05;

% number of preview points
n=input('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 %%%%%%%%%%

% road shift operators

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

%D=[0 1 0 0 0 0 0 0 0 0 ;
%0 0 1 0 0 0 0 0 0 0 ;
%0 0 0 1 0 0 0 0 0 0 ;
%0 0 0 0 1 0 0 0 0 0 ;
%0 0 0 0 0 1 0 0 0 0 ;
%0 0 0 0 0 0 1 0 0 0 ;
%0 0 0 0 0 0 0 1 0 0 ;
%0 0 0 0 0 0 0 0 1 0 ;
%0 0 0 0 0 0 0 0 0 1 ;
%0 0 0 0 0 0 0 0 0 0];
%E=[0;0;0;0;0;0;0;0;1];

%%%%%%%%%%% Linear Car Model %%%%%%%%%%%

Linear_car_model
disp('')

%%%%%%%%%%% Linear Control Gain Calculation %%%%%%%%%%%
```

```
% 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)
% We keep the same priorites.
R1cost=R1;
R2cost=R2;

tic %start a stoopwatch timer
disp('     Loading path information......')

%%%%%%%%%%%% Path Information %%%%%%%%%%%%

for epoch = 1:5
   if epoch ==1
   circuits_2
   else
   circuit_iterations
   end

   [K,nb] = size (yref) %array size of yref

%%%%%%%%%%%% State Definition & Initialisation %%%%%%%%%%%%

% At each time 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 ]

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 matrix
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+5 (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 the local x and y axes of the car.

% definition of the state 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 is changed
ZinitB(2) = ZinitB(2)-u*sin((phiB(k)-phiB(k-1)));
%ZinitC(2) = ZinitC(2)-u*sin((phiC(k)-phiC(k-1)));  %the local y-axis is changed
```

```
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 remainning 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 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
if dJdw/prevdJdw < 1.000  %cost ratio
gama=1.05*gama;
end
if dJdw/prevdJdw > 1.005
gama=0.7*gama;
end
% difference calculation Polak Ribiere or Gradient method
% prevdeltaw=deltaw;
% if k<3
deltaw=-gama*dJdw; %value for deltaw
gama_next(k+1)=gama;

% else
% deltaw=-gama*(dJdw - 1/norm(prevdJdw) *...
% (dJdw'*(dJdw-prevdJdw)*prevdeltaw')');
% end

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

%%%%%%%%%%%%% End of Weighting 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(4)
figure(2)
plot(jg)
            %plot(gama)
xlabel('No. of Epoch');
ylabel('Learning rate');
title ('Plot of Learning Rate')
grid on

% figure(5)
figure(3)
ww1=W_end(:,10)'
ww=[W_init(10),ww1];

% figure(6)
figure(3)
plot(ww)
xlabel('No. of Epoch');
ylabel('Weight');
title('Plot of Updated Weight vs No. of Epoch')
grid on

Plottings2inoneshot
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Linear Car Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Linear_Car_Model.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Linear Car Model
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%continous state space form of a car model

%global frame

Acar = [  0     1              0                0     ;
        0  -(Cf+Cr)/(M*u)    (Cf+Cr)/M      (b*Cr-a*Cf)/(M*u);
        0     0              0                1     ;
        0  (b*Cr-a*Cf)/(Iz*u)  (a*Cf-b*Cr)/Iz  -(a^2*Cf+b^2*Cr)/(Iz*u);]

Bcar = [0  ;
     Cf/(M*G);
     0;
     a*Cf/(Iz*G);];

Ccar = [1 0 0 0];
Dcar = [0];

%discrete state space form of a car model

[Ad Bd] = C2D(Acar, Bcar, T);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Linear Quadratic Regulator (LQR)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LQRgain.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Linear Quadratic Regulator (LQR)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Compute the Linear control gain obtained with the LQR theory

  C=[ 1 0 0 0  -1      0    zeros(1,n-1) ;
     0 0 1 0 1/(u*T) -1/(u*T)  zeros(1,n-1)  ];

  R1=C'*Q*C;

  A=[ Ad          zeros(4,n+1);           % Car model
     zeros(n+1,4)    D      ];            % Road model
  B=[   Bd;
    zeros(n+1,1)];
```

```
%non-preview gain using the DLRMI function
%We could directly have used [Kt,Sbis,Ebis] = DLQR(A,B,R1,R2)
%but the time to compute would be much greater

  [K1,P11] = dlqrmi(Ad,Bd,R1(1:4,1:4),R2);    % K1 is gain matrix : feedback gain
                                              % P11 is Riccati equation solution

% Use non-preview results to solve the preview problem.

  FC = Ad - Bd*K1;          % xdot

  P12 = zeros(4,n+1);
  P12(:,1) = R1(1:4,4+1);     % replaces the whole row and first column of P12 with first four rows
                              % and fifth column of R1

  for i=2:n+1
    P12(:,i) = R1(1:4, 4+i) + FC'*P12(:,i-1);
  end

  K2 = inv(Bd'*P11*Bd + R2) * Bd' * P12 * D;

  Kt=[K1 K2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Road Models 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% circuits_2.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Road Models
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


disp('Type of circuit ?')
disp('   1  Sinus Shape ')
disp('   2  Lane Change ')
disp('   3  Sudden Change of Direction')
disp('   4  Smooth Random Path')
disp('   5  Obstacle Course')

ii=input(' ');

if ii==1
% sinus shape
xref=[0:u*T:900];
yref=50*sin(xref/100);
end

if ii==2
%lane change
xr1=[0:u*T:50-u*T];
xr2=[50:u*T:50+60];
xr3=[110+u*T:u*T:300];
xref=[ xr1 xr2 xr3];
yr1=0*xr1;
yr2=2-2*cos((pi/60)*xr2-50*pi/60);
yr3=4*ones(size(xr3));
yref=[yr1 yr2 yr3];
end

if ii==3
```

```
%sudden change of direction
xr1=[0:u*T:60-u*T];
xr2=[60:u*T:200];
xref=[xr1 xr2];
yr1 = 0*xr1;
SIZE_OF_xr2=size(xr2);
yr2 = 0.5*0.25*[1:SIZE_OF_xr2(1,2)]*0.5359';
yref = [yr1 yr2];
end

if ii==4
%smooth random path
K=900/(u*T)+1;
xref=[0:u*T:900];
[Bfilter, Afilter] = butter(5, 0.007);
roadn=10*rand(K,1);
roadn = 40*(roadn-5);
yref = filter(Bfilter, Afilter, roadn)';
end

if ii==5
%obstacle course path
load dirResult
xref = x(1,:);
yref(1,:) = x(2,:);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Road Models 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% circuit_iterations.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Road Models
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%if ii==1
% circle circuit
%R=250;
%xr1=[R:-u*T:-R]; %
%xr2=[-R+u*T:u*T:R];
%xref=[xr1 xr2];
%yr1=sqrt(R^2-xr1.^2);
%yr2=-sqrt(R^2-xr2.^2);
%yref=[yr1 yr2];
%end

if ii==1
% sinus shape
xref=[0:u*T:900];
yref=50*sin(xref/100);
end

if ii==2
%lane change
xr1=[0:u*T:50-u*T];
xr2=[50:u*T:50+60];
xr3=[110+u*T:u*T:300];
xref=[ xr1 xr2 xr3];
yr1=0*xr1;
yr2=2-2*cos((pi/60)*xr2-50*pi/60);
```

```
yr3=4*ones(size(xr3));
yref=[yr1 yr2 yr3];
end

if ii==3
%sudden change of direction
xr1=[0:u*T:60-u*T];
xr2=[60:u*T:200];
xref=[xr1 xr2];
yr1 = 0*xr1;
SIZE_OF_xr2=size(xr2);
yr2 = 0.5*0.25*[1:SIZE_OF_xr2(1,2)]*0.5359';
yref = [yr1 yr2];
end

if ii==4
%smooth random path
K=900/(u*T)+1;
xref=[0:u*T:900];
[Bfilter, Afilter] = butter(5, 0.007);
roadn=10*rand(K,1);
roadn = 40*(roadn-5);
yref = filter(Bfilter, Afilter, roadn)';
end

if ii==5
%obstacle course path
load dirResult
xref = x(1,:);
yref(1,:) = x(2,:);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Graphs Display

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plottings2inoneshot.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Graphs Display
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


road_angle_estimation=zeros([1:K-n+1],1);
for j=1:K-n
   road_angle_estimation(j)=(yref(j+1)-yref(j))/(u*T);
end
yaw_errorA=phiA([1:K-n])-road_angle_estimation([1:K-n]);

figure(1)
%path following
subplot(2,1,1)
plot(xref([1:K-n]),yref(1,[1:K-n]),xref([1:K-n]),global_positionA(1:K-n),'g:');
%plot(xglobal([1:K-n+1]),yref(1,[1:K-n+1]),xglobal([1:K-n+1]),yglobal(1:K-n+1,1),'g');grid;
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('y coordinate , m');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
h = get(gca, 'title');
```

```
set(h, 'FontSize', 14);
   if i==2
      AXIS([-10 1000 -100 100])
   end
hold on;

% lateral velocity
subplot(2,1,2)
plot(xref([1:K-n]),lateral_accelerationA(1:K-n),':');
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);

ylabel('lateral acceleration, m/s^2');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Lateral Acceleration at Mass Center m/s^2');
h = get(gca, 'title');
set(h, 'FontSize', 14);
hold on;

figure(2)
%y difference
subplot(2,1,1)
plot(xref([1:K-n]),yref([1:K-n])-global_positionA([1:K-n]),':');
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('y error , m ');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Y Path Following Error');
h = get(gca, 'title');
set(h, 'FontSize', 14);
hold on;

%yaw error
subplot(2,1,2)
plot(xref([1:K-n]),yaw_errorA',':');
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('yaw angle error');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Yaw Attitude Angle Error');
h = get(gca, 'title');
set(h, 'FontSize', 14);
hold on;

figure(3)
%steering wheel angle
subplot(2,1,1)
plot(xref([1:K-n-1]),deltaA(1:K-n-1),':');
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('steering wheel angle , rad ');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Steering Wheel Angle');
h = get(gca, 'title');
set(h, 'FontSize', 14);
hold on;

%attitude angle
subplot(2,1,2)
```

```
plot(xref([1:K-n]),road_angle_estimation(1:K-n),xref([1:K-n]),phiA(1:K-n),'g:');
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('attitude angle , rad ');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Attitude Angle Following');
h = get(gca, 'title');
set(h, 'FontSize', 14);
hold on;

yaw_errorB=phiB([1:K-n])-road_angle_estimation([1:K-n]);

figure(1)
%figure(4) %br tambah
%path following
subplot(2,1,1)
plot(xref([1:K-n]),yref(1,[1:K-n]),xref([1:K-n]),global_positionB(1:K-n),'g');grid
%plot(xglobal([1:K-n+1]),yref(1,[1:K-n+1]),xglobal([1:K-n+1]),yglobal(1:K-n+1,1),'g');grid;
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('y coordinate , m');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
h = get(gca, 'title');
set(h, 'FontSize', 14);
   if i==2
      AXIS([-10 1000 -100 100])
   end
hold on;

% lateral velocity
subplot(2,1,2)
plot(xref([1:K-n]),lateral_accelerationB(1:K-n));grid;
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);

ylabel('lateral acceleration, m/s^2');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Lateral Acceleration at Mass Center m/s^2');
h = get(gca, 'title');
set(h, 'FontSize', 14);

figure(2)
%figure(5) %br tambah
%y difference
subplot(2,1,1)
plot(xref([1:K-n]),yref([1:K-n])-global_positionB([1:K-n]));grid;
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('y error , m ');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Y Path Following Error');
h = get(gca, 'title');
set(h, 'FontSize', 14);
hold on;

%yaw error
subplot(2,1,2)
plot(xref([1:K-n]),yaw_errorB');grid;
xlabel('distance, m');
```

```
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('yaw angle error');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Yaw Attitude Angle Error');
h = get(gca, 'title');
set(h, 'FontSize', 14);

hold on;

figure(3)
%figure(6) baru tambah0
%steering wheel angle
subplot(2,1,1)
plot(xref([1:K-n-1]),deltaB(1:K-n-1));grid;
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('steering wheel angle , rad ');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Steering Wheel Angle');
h = get(gca, 'title');
set(h, 'FontSize', 14);
hold on;

%attitude angle
subplot(2,1,2)
plot(xref([1:K-n]),road_angle_estimation(1:K-n),xref([1:K-n]),phiB(1:K-n),'g');grid;
xlabel('distance, m');
h = get(gca, 'xlabel');
set(h, 'FontSize', 14);
ylabel('attitude angle , rad ');
h = get(gca, 'ylabel');
set(h, 'FontSize', 14);
title('Path Following');
title('Attitude Angel Following');
h = get(gca, 'title');
set(h, 'FontSize', 14);

hold on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Y Path Following Error & Steering Wheel Angle Display

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% avg.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Y Path Following Error & Steering Wheel Angle Display
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if size(yref) == size(global_positionA)
   countLimit = K-n;
else
   countLimit = min(size(yref),size(global_positionA));
end
myCount = 1;
```

```
averageError = 0;
totalError = 0;
thisError = 0;
while myCount < countLimit(2)
   thisError = yref(myCount) - global_positionA(myCount);
   if thisError < 0
      thisError = -thisError;
   end
   totalError = totalError + thisError;
   myCount = myCount + 1;
end
averageError = totalError/myCount;

averageError

max_angle=max(deltaA)

min_angle=min(deltaA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```