Hardware Implementation of Soft Computing Approaches for an Intelligent Wall-following Vehicle

by

Willie Tsui

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2007

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Abstract

Soft computing techniques are generally well-suited for vehicular control systems that are usually modeled by highly nonlinear differential equations and working in unstructured environment. To demonstrate their applicability, two intelligent controllers based upon fuzzy logic theories and neural network paradigms are designed for performing a wall-following task and an autonomous parking task. Based on performance and flexibility considerations, the two controllers are implemented onto a reconfigurable hardware platform, namely a Field Programmable Gate Array (FPGA). As the number of comparative studies of these two embedded controllers designed for the same application is limited in the literature, one of the main goals of this research work has been to evaluate and compare the two controllers in terms of hardware resource requirements, operational speeds and trajectory tracking errors in following different pre-defined trajectories. The main advantages and disadvantages of each of the controllers are presented and discussed in details. Challenging issues for implementation of the controllers on the FPGA platform are also highlighted. As the two controllers exhibit benefits and drawbacks under different circumstances, this research suggests as well a hybrid controller scheme as an attempt to integrate the benefits of both control units. To evaluate its performance, the hybrid controller is tested on the same pre-defined trajectories and the corresponding results are compared to that of the fuzzy logic and the neural network based controllers. For further demonstration of the capabilities of the wall-following controllers in other applications, the fuzzy logic and the neural network controllers are used in a parallel parking system. We see this work to be a stepping stone for further research work aiming at real world implementation of the controllers on Application Specified Integrated Circuit (ASIC) type of environment.

Acknowledgements

The author would like to thank her supervisor, Dr. Fakhreddine Karray, for his guidance and support on this research work. The author would also like to acknowledge Dr. Insop Song for his advice, and Keith Gowan, Jason Nery, Henrick Han and Tony Sheng for their contributions in this project. Furthermore, the author would like to thank Dr. William Melek and Dr. Otman Basir for reviewing, and Razim Samy for editing the thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Introduction

Modern systems are becoming increasingly more complex, making conventional control algorithms that utilize mathematical models insufficient. This resulted in the evolution of soft computing techniques based on biological processes such as learning and evolutionary development. Soft computing techniques model human intelligence, providing decisions given imprecise and uncertain information. The three most commonly employed soft computing tools are fuzzy logic, neural networks and genetic algorithms. Fuzzy logic reasoning is based on human experience and expert knowledge. It has the ability to generate a decision given vague and imprecise information. Neural network reasoning depends on the extraction of hidden relationships in given datasets. It has the ability to learn from examples, drawing conclusions based on past experiences. Genetic algorithm is essentially an optimization technique based on the ideas of evolution in biological development. It has the ability to systematically obtain solutions in complex problems.

Soft computing techniques have been ever-increasing in importance, and have wide applications in industries. For example, General Electric developed smart washing machines that sense the weight of each load and water temperature to determine the best water level, wash time and laundry method [1]. They are designed to minimize energy requirements and reduce the chances of wear and tear in clothing. Another example is the use of artificial intelligence methodologies in rice cookers manufactured by Zojirushi [2]. These cookers automatically adjust the heating temperature and time to ensure the cooked rice has reached the desired texture. Other applications in household appliances include intelligent air conditioners, cameras, dishwashers, and refrigerators. Aside from appliances, soft computing methodologies are also being applied to other industries. In the financial sector, Hecht-Nielsen Neurocomputer Corporation developed the Falcon software for real-time debit and credit card fraud detection [3]. The system learns an individual's purchasing behavior based on transaction history and signals any unusual transaction, providing better security to consumers, merchants and financial institutions. As it is better at predicting consumer purchasing patterns, the amount of unnecessary intrusions are greatly reduced. Other examples utilizing soft computing technology include intelligent marketing systems in the financial sector, smart medical diagnostic systems in the biomedical industry, and automated anti-lock braking and cruise control systems in the automotive industry.

1.2 Motivation

One of the most widely researched topics in the soft computing community is autonomous vehicles. The term autonomous represents the elimination of human control. Thus, autonomous vehicles

generally refer to vehicular systems capable of performing navigation tasks without human guidance. Autonomous vehicles are a promising technology with the ability for reduced traffic congestion and decreased accidental rates. Common tasks which require automation include lane-following, parking, collision avoidance, vehicle following, traffic signal detection, and navigation at intersections. Of special interest is lane-following and parallel parking, as these are frequently performed tasks that can be easily automated. Their study is often generalized by investigating wall-following, due to the similarities amongst all three.

1.2.1 Lane-following versus Wall-following

Lane-following can be separated into two steps: lane detection and lane tracking. The first relates to the accurate location of the lane boundaries while the latter deals with maintaining the vehicle within the lane boundaries. This is much similar to the wall-following task in which the goal is to detect the distance between the vehicle and the wall and to maintain a predefined distance from the wall. The main difference between the two tasks lie in the method for path detection. Lane-following relies upon the detection of marked lane boundaries located on the ground while wall-following requires the measurement of the relative distance to a side wall.

1.2.2 Parallel Parking versus Wall-following

The similarity between parallel parking and wall-following lies in the perception mechanism. Parallel parking requires the detection of other parked vehicles to determine the availability of a suitable parking space. Similar to wall-following, the vehicle is also required to follow the curb during parking maneuvers. However, the major differences between parallel parking and wall-following lie in the maneuvering algorithm and detection distance. In wall-following, the vehicle is consistently moving uni-directionally (forward or backward) for an extended duration (typically for the whole length of the path) at a constant pre-defined distance from the wall. In contrast, the vehicle behaves differently in the parallel parking task. To perform parallel parking, the vehicle performs wall-following as it searches for a parking space, and with the gathered data, logically decides whether any detected spaces are sufficient in size for the vehicle to park into. During the parking maneuvers, the autonomous vehicle may spontaneously reverse its direction of travel. The switch in direction depends on the successful detection of a valid parking space, upon which the vehicle would reverse into the detected space. Also, depending on the presence of a potential parking space, the tracking distance differs. Short distance tracking is necessary when parked cars are in close proximity to the autonomous vehicle. The vehicle follows the ‼ wall¶ of parked cars and the sensors would utilize the cars for distance measurements. However, far distance tracking is required when a potential parking space is detected. In this scenario, the vehicle must depend on the curb for distance measurements. Due to its applicability to lane-following and parallel parking and its ease of implementation, the wall-following task is chosen for study.

Traditionally, control of vehicular systems is realized using conventional approaches, of which the most well known is the Proportional-Integral-Derivative controller. However, classical approaches generally require the use of complex mathematical models, making the design of controllers difficult. With the advancements in soft computing and computational intelligence, many designers have investigated the feasibility of utilizing soft computing techniques for vehicular system control applications. Some focused on fuzzy logic control as it is well-suited for nonlinear and complex systems where human knowledge and experiences can be drawn upon. For vehicular control, human expert knowledge is readily available as human operators can perform driving tasks, such as lane-following and vehicle following, with ease. A driver will instinctively steer to adjust its heading if a vehicle drifts into the adjacent lane. In the same manner, a driver will speed up or slow down depending on their relative distance to the vehicle ahead. This information justifies the use of fuzzy logic to mimic human driving behaviors. Aside from fuzzy logic control, engineers also investigated the suitability of neural network control. Neural network control is well-suited for complex systems where there is an abundance of experimental data. As human drivers can operate vehicles to perform driving tasks with ease, experimental data can be collected during actual driving experience, confirming the sutiability of neural network control.

The application of fuzzy logic and neural network control units for autonomous vehicular control and in particular for wall-following has been widely reported in literature. Software-based platforms such as microcontrollers are often utilzed for the realization of such controllers. However, with growing demand for speed, implementations on hardware platforms featuring high performance are preferred. Aside from standalone fuzzy logic or neural network based controllers, many researchers also investigate the feasibility of combining the two soft computing techniques. However, only a few attempts have been made to compare the two controllers under the same environment [4, 5, 6]. In this research, a fuzzy logic and a neural network controller are implemented on a hardware-based platform, namely a Field Programmable Gate Array (FPGA), and their corresponding performance when presented with the same environmental settings are compared.

1.3 Objectives

The main objectives of this research work can be summarized as follows:

- Develop an intelligent wall-following system based on fuzzy logic theories and neural network paradigms.
- Realize the main control unit on a reconfigurable hardware unit (FPGA) for high performances.
- Perform a comparative study on the two intelligent controllers in terms of their hardware resource requirements, operational speeds and trajectory tracking errors. Outline the major advantages and drawbacks in utilizing each of the two.

- Develop a hybrid controller with a switching mechanism to select between the fuzzy logic and the neural network controller under different circumstances, with the goal of attaining the benefits of both control units.
- Apply the developed wall-following system experimentally in parallel parking applications.

1.4 Contributions

The main contributions of this research work can be summarized as follows:

- The wall-following performance of a neural network controller is compared to that of a fuzzy logic control unit. Major advantages and drawbacks in utilizing each of the two controllers are identified. In general, hardware implementation of the fuzzy logic controller requires less hardware resources, with smaller rise and fall times. Hardware implementation of the neural network controller provides higher bandwidth while offering smaller trajectory tracking errors.
- A hybrid controller is designed to combine the benefits of the fuzzy logic and the neural network control units. At each point in time, a switching mechanism selects either the fuzzy logic controller or the neural network controller based on the vehicle's actual distance with respect to the predefined wall-following distance. Performance evaluation of the hybrid controller is provided. Overall, this controller attains the fast response of the fuzzy logic controller. At the same time, it inherits the small trajectory tracking error feature of the neural network controller.
- A potential application of the wall-following system is investigated. Both the fuzzy logic and the neural network controller are proven successful in performing the parallel parking task.

1.5 Thesis Organization

This thesis is organized as follows:

- Chapter 2 presents a review of some earlier attempts in the realization of autonomous wall-following, lane-following and parallel parking systems. Both conventional and soft computing approaches are presented. In addition, prototyping platforms are investigated.
- Chapter 3 provides background information on various topics including fuzzy logic theories, neural network concepts, reconfigurable hardware design and vehicular modeling.
- Chapter 4 describes the hardware design and implementation aspect of the prototype vehicle, developed for testing different control units.
- Chapter 5 provides details regarding the design and implementation of two controllers based on different soft computing paradigms, namely fuzzy logic and neural networks. An extensive comparison of the two control units is performed. Three metrics for their

evaluations include computational resource requirements, operational speeds and trajectory tracking errors. Major advantages and drawbacks of each controller are identified.

- Chapter 6 presents a hybrid controller design, with combined benefits of the fuzzy logic and the neural network controllers. It also illustrates an application of the intelligent wall-following system, in which the system is proven applicable in performing parallel parking.
- Chapter 7 concludes the research work with suggestions for future studies.

Chapter 2
Literature Review

This chapter provides a detailed literature survey of previous works on the implementation of wall-following, lane-following and parallel parking systems. Both conventional and soft computing approaches are presented. A review on different prototyping platforms is also given.

2.1 Conventional Approaches

Early attempts in the field of autonomous vehicular control focused on classical control techniques. Murray and Sastry [7] proposed the use of sinusoidal control inputs to steer nonholonomic systems, including vehicular systems, from an initial configuration to a final configuration. They demonstrated that vehicles can follow different paths by means of sinusoidal inputs. Based on this result, many later researches segmented control problems related to autonomous vehicles into two sub-problems: path planning and motion tracking. The goal of the former is to plan a feasible path for realization of the control task while the goal of the latter is to steer the vehicle along the designed path by means of sinusoidal control inputs. One example of such research is the development of the autonomous parallel parking system by Paromtchik and Laugier [8]. The parking system is divided into three steps: detection of a parking space, vehicle adjustment to starting position and parking maneuver. In the last step, the vehicle iteratively plans for the next maneuver and utilizes sinusoidal control inputs to follow the planned trajectory for reversal into the parking space.

The research of Kanayama *et al.* [9] took a different route, in which a state feedback controller is utilized to perform path tracking. An error posture is defined as the difference between the path to be followed and the current posture of the vehicle. The goal of the controller is to minimize this posture, ideally to zero, by modifying the steering angle according to the error configuration. Experimental results illustrate the effectiveness of the proposed controller. Egersdet *et al.* [10] presented a similar approach for performing path following. Once again, the error posture is calculated as the difference between the current posture and the reference posture. In addition, the steering angle is set to be proportional to the difference. However, in this research, the desired configuration depends not only on the desired trajectory, but also on the current vehicle configuration. If the difference between the desired and current positions exceeds a predefined limit, the reference posture will remain unchanged in the next time point instead of moving along the designed trajectory. This approach is labeled as the virtual vehicle approach and has the ability to ensure that the path tracked by the vehicle is smooth. In [11] and [12], the authors employed a linear-parabolic model and a sliding mode controller for lane-following and trajectory tracking purposes.

2.2 Soft Computing Methodologies

Conventional approaches are prone to many issues such as difficulties in modeling uncertainties in the real-world environment and processing of noisy sensory information. In addition, the large number of nonlinearities and coupling dynamics in vehicular models complicate the design of classical controllers. This encourages the application of soft computing techniques, such as fuzzy logic and neural networks, to algorithms utilized in autonomous vehicular systems, eliminating the necessity of complex mathematical models.

Antonelli *et al.* [13] presented a fuzzy logic based controller for mobile robot lane or path tracking. A vision-based system is employed to extract the two input variables, curve and distance, of the proposed controller. Simple rules based on linguistic terms are defined to generate the linear and angular velocities of a vehicle for path following purposes. Li *et al.* [14] designed a fuzzy controller for parking control of a car-like mobile robot. The proposed controller realizes both garage parking and parallel parking maneuvers. Two input variables representing the desired and the current orientations of the vehicle is employed for generation of the output steering angle. To determine the current position and orientation, the prototype is equipped with a vision system. Sensory information is sent through a wireless radio link to a host computer, in which image processing and desired motion signal generation is performed. Utilizing the same wireless radio link, the vehicle receives and executes the motion commands. Chao *et al.* [15] proposed a two-step intelligent parallel parking system. In the first step, the system utilized information from an omni-directional vision system to generate the parallel parking trajectory. A fuzzy logic controller is then employed to track the reference trajectory by controlling the steering angle of the vehicular system. This technique is claimed to be advantageous due to its soft switching behavior and robust performance. Rahman and Jafri [16] proposed a behaviour based navigation system with a two layered architecture. Utilizing sensory information, the first layer selects one or more of the following behaviors: obstacle avoidance, corridor following, wall-following, target steering and emergency. Fuzzy logic controllers are employed to determine the left and right wheel velocities for each selected behavior. The wheel velocities are then coordinated with distance measurements collected from six ultrasonic range sensors. In [17, 18, 19], the authors applied conventional fuzzy logic controllers and fuzzy gain scheduling controllers for wall-following, goal seeking and parallel parking purposes.

Yang and Meng [20] took a different approach in solving the parking problem. They presented a real-time collision free motion planning algorithm using a neural network based approach. Unlike conventional neural network models, dynamics of each neuron is characterized by a shunting or additive equation derived from Hodgkin and Huxley's [21] membrane equation. Weights are predefined at the stage of neural network design, eliminating the need for a learning procedure. The algorithm demonstrated effectiveness in motion planning for parallel parking maneuvers, navigations in a complex house-like and dynamically changing environments. Zalama *et al.* [22] designed a neural network model for navigation. Reinforcement learning is employed in which the control system learns the appropriate action through a credit and punishment system that assigns rewards and penalties according to the correctness of the control action. The goal is to maximize the long term

reward. Based on this technique, the control system successfully learnt collision avoidance, wall-following and goal reaching behaviors. Cicirelli *et al.* [23] targeted the wall-following behavior with the neural Q-learning control algorithm, which is a subset of reinforcement learning. The Q value is defined as the expected return when performing a specific action at a particular state. With this information, given any current state, the optimal action can be taken as that with the highest Q value. To learn the Q values or to perform Q-learning, feed-forward neural networks with one hidden layer and a single output are designed. Rewards and penalties received after execution of each action is used for training of the networks. Based on experimental data, the wall follower is proven successful in following the pre-defined wall layout. In [24, 25, 26], the authors applied neural network based algorithms to lane-following, navigation and motion planning problems. These methodologies are also applicable to wall-following and parking maneuvers.

The major challenges in designing effective fuzzy controllers lie in the generation of representative fuzzy membership functions and fuzzy if-then rules while the main issue with neural network controller designs is the lack of linguistic knowledge representation. To address these issues, many researchers proposed the use of hybrid algorithms combining the advantages of fuzzy logic and neural networks. Li *et al.* [27] presented a neural-fuzzy inference network to realize the wall-following task. In the proposed design, variables of the fuzzy system, such as fuzzy membership functions and fuzzy rules, are optimized online by a neural network trained from training patterns generated by human operators. Simulation results illustrate the robustness of the proposed system. Utilizing only sensory information, the vehicle is steered to follow different wall setups, including those that are significantly different from the walls used in training. Khoshnejad and Demirli [28] suggested a neuro-fuzzy behavior based controller for performing parallel parking in a single reverse maneuver. To mimic behavior of human drivers, the membership functions are tuned by a neural network trained from training data acquired from experiences of human drivers. For simulation purpose, the reference parallel parking trajectory is assumed to be a fifth-order polynomial. The final vehicular system, incorporated with the neuro-fuzzy controller, is deemed successful as it is able to parallel park into spaces of various sizes. In [29, 30, 31], the authors proposed other hybrid techniques, combining the theories of fuzzy logic and neural networks for realization of parking and wall-following controllers.

The use of a fuzzy logic controller and a neural network controller, either separately or cooperatively, is widely seen in literature. However, comparative studies on the two types of controllers for the same vehicular application are limited. Nijhuis *et al.* [4] provided an evaluation of a fuzzy logic and a neural network controller for the purpose of obstacle avoidance. The two controllers are compared in terms of their abilities in handling complex sensory data, requirements of expert knowledge, comprehensibility in knowledge representation and ease of hardware implementation. It is concluded that the development phase of the neural network controller is simpler as the control unit can extract hidden relationships among information from different sensors, while the fuzzy logic controller requires explicit transformation of the data into linguistic variables. In terms of dependency on human knowledge, the neural network controller requires a set of representative training data, which determines the controller quality. The fuzzy logic controller also

requires expert knowledge for definition of linguistic variables, fuzzy rules and membership functions. However, the transformation of knowledge into these parameters pose a challenge as there is currently no standard methodology for obtaining this transformation. As explicit rules are presented in the fuzzy logic controller, it can be easily interpreted. In comparison, representing knowledge in a set of mathematical equations makes the internal structure of the neural network controller difficult to understand. Godjevac [5] also provided a comparative study of fuzzy logic control and neural network control on the obstacle avoidance application. Similar conclusions were made including the difficulty in transforming expert knowledge into fuzzy logic controller parameters and the high comprehensibility of fuzzy logic controllers in representing knowledge as linguistic rules. Other observations include the long computational time in fuzzy controllers due to the need for complex fuzzification and defuzzification operations and the difficulty in understanding the internal structure of neural network controllers as knowledge is incorporated into numerical weights. Almeida *et al.* [6] developed a simulator for comparison of the performance of different control strategies in performing the lane-following task. It is proposed that fuzzy controllers are easy to tune for performance improvements, and neural network controllers are powerful but with certain disadvantages such as a time consuming training phase and a computationally expensive implementation. The above reviewed methodologies are based on fuzzy logic and neural networks due to their relevance in this thesis. However, soft computing approaches applied in autonomous vehicles are not limited to those studied. Proposals related to other techniques can be found in [32, 33, 34, 35].

2.3 Prototyping Platform

To demonstrate feasibility, researchers often present simulation results. However, due to various differences between simulated and real-world environments, prototypes are frequently developed to facilitate research. Implementations of algorithms to prototypes generally make use of one of the following platforms: General-Purpose Processors (GPPs), Application Specified Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs). Characterized by flexibility, many proposed algorithms are implemented on software-based GPPs. Peri and Simon [36] utilized two Microchip PIC16F877 microcontrollers, programmed in C, for motion control of a robot along a predefined trajectory. Information from various sensors is first sent to one of the two microcontrollers for determination of the appropriate wheel velocities based on fuzzy logic. These signals are then transmitted to another microcontroller to generate the corresponding control signals for servo motor control. Divelbiss and Wen [37] proposed a trajectory tracking and parallel parking algorithm for a car-trailer system, in which an on-board computer is utilized for motor control. The prototype vehicle is equipped with seven optical encoders to determine the relative position of the vehicle and the trailer. This information is sent to the on-board computer programmed in C++, which serves as the main control unit for the generation of steering and motion control signals. Jiang [38] also targeted the parallel parking problem. The steering and motion control mechanism is realized on a notebook computer attached to the prototype system. Utilizing information from ultrasonic sensors, the designed autonomous vehicular system successfully parked into a small parking space. GPPs are also the selected platform in [14], [17], [22] and [29].

With growing demand for speed and performance, researchers also looked into hardware FPGA-based approaches. Compared to GPPs, FPGAs provide potentially higher performance with minimal sacrifice in flexibility. Kim [39] implemented a reconfigurable FPGA-based fuzzy logic controller to back a truck into a loading dock as quickly and precisely as possible. Experimental results illustrate that the proposed implementation not only satisfies the design goal of backing into the loading dock, it also demonstrates the increase in performance when compared to C language based software control systems. Li *et al.* [40] used a FPGA to control the steering angle and velocity of a car-like robot for realization of a wall-following and parking system. The presented hardware architecture utilizes six infrared sensors to detect environmental settings. Sensory information is then passed to an FPGA chip, where the fuzzy logic controller is located. Connected to the FPGA board, the servo motors collect and execute the motion control signals for the realization of different vehicular maneuvers. Bellis *et al.* [41] developed a wall follower on the FPGA platform. The main control unit is realized by neural networks and is implemented in hardware coding. To ensure accuracy, a simulation tool is developed. Outputs generated by the FPGA-based controller are verified against the simulated results.

2.4 Summary

Early attempts on realization of wall-following, lane-following and parallel parking systems make use of controllers that perform path planning and path tracking. A feasible path is first designed and the vehicle is then controlled to steer along the planned path using sinusoidal control inputs, state feedback approaches or virtual vehicle techniques. With fast development in soft computing and artificial intelligence, approaches based on fuzzy logic and neural network paradigms have been increasing in acceptance. These methodologies eliminate the use of complex mathematical models, thus shortening the development time. However, application specific comparative studies of fuzzy logic and neural network controllers remain limited in literature. In terms of prototyping platform, earlier attempts often made use of General-Purpose Processors due to their flexibility. However, some recent implementations focus on reconfigurable hardware due to its superiority in performance.

Chapter 3
Background

This chapter introduces fundamental theories of soft computing tools. The introduction is restricted to the areas of fuzzy logic and neural networks due to their relevance in this research work. Background information on reconfigurable hardware design and conventional vehicle dynamics is also presented in this chapter.

3.1 Basic Concepts in Soft Computing

3.1.1 Fuzzy Logic

Conventionally, industrial reasoning processes are based on crisp binary logic, in which variables can only assume one of two values: True and False. However, human knowledge is often represented by qualitative and subjective variables, making direct application of human experience or knowledge into industrial process control difficult. To take advantage of human reasoning for processing of imprecise and uncertain information, Zadeh proposed the concept of fuzzy sets and fuzzy logic to mimic the human decision-making process [42]. In general, instead of representing a system as a set of complex mathematical equations, fuzzy systems use simple empirical rules to represent input and output relationships. The suitability of fuzzy control does not depend much on the number of inputs and outputs, but rather depends on the availability of human expert knowledge.

Fuzzy Sets

The universe of discourse, defined as the largest set that one could consider in a particular problem domain, contains all possible elements a variable can attain. It is impossible for a variable to assume a value outside of its universe of discourse. For example, let A be a crisp set in the universe of discourse X, then in crisp logic, an element x inside A is defined as a member of A while an element outside of A is not a member of A. An element can only be a member or a non-member of A. An example of a crisp set is the choice of gear available for a five-gear manual transmission. In mathematical terms, the set can be denoted as A = {P, R, 1, 2, 3, 4, 5}. Representing human knowledge by crisp sets is a challenge as human experience is often descriptive. For example, a set with all “high” speeds cannot be represented by a crisp set. The term “high” is a linguistic variable that is qualitative, subjective and fuzzy, often termed as a fuzzy descriptor. An individual may consider speeds above 80 km/h as high while others may consider speeds above 100 km/h as high.

To deal with linguistic fuzzy terms in fuzzy logic, a fuzzy descriptor is characterized by a membership function, which maps elements from the universe of discourse to numerical values

between zero and one for representation of the grade of membership. Let A be a fuzzy set in the universe of discourse X, then similar to crisp logic, an element with membership grade of one represents a member of A while an element with membership grade of zero is not a member of A. However, in fuzzy logic, variables can also take on a membership grade greater than 0 and less than 1 to illustrate partial membership. In other words, a variable can be associated to a certain degree as being within the set A, while also being outside the set. In mathematical notation, a fuzzy set A can be represented by $A = \{(x, \mu_A(x));\ x \in X,\ \mu_A(x) \in [0,1]\}$ where X is the universe of discourse with elements x and $\mu_A(x)$ is the membership function with membership grades ranging from 0 to 1. For a set of variables, membership functions can assume different shapes, ranging from the simplest ones formed by straight lines to others made up of nonlinear curves. The most commonly employed membership functions are of the triangular, trapezoidal and Gaussian types [43].

Fuzzy Operators

In fuzzy logic, human expert knowledge is translated to if-then rule sets that comprise fuzzy sets and fuzzy operators. For example, the following fuzzy rule consists of the AND operator and fuzzy sets near, fast and high.

IF obstacle is *near* AND speed is *fast* THEN deceleration is *high*

The two most commonly utilized fuzzy operators are the AND operator and the OR operator, which are generally referred as the T-norm operator and the S-norm operator. The definitions of these operators are not unique. With satisfaction of the corresponding boundary, monotonicity, commutativity and associativity conditions, any operator can be classed as a T-norm or S-norm operator [43]. One example of the T-norm and the S-norm operators are the minimum operator and the maximum operator, respectively. Denoted as min(a,b), the minimum operator computes the lower membership grade of a and b for each element in the universe of discourse. In the same manner, the maximum operator calculates the larger membership value of a and b and is denoted as max(a,b).

Fuzzy Inference Systems

Fuzzy inference systems consist of three major components: fuzzifier, inference engine and defuzzifier. Figure 1 illustrates the basic configuration of a fuzzy inference system.

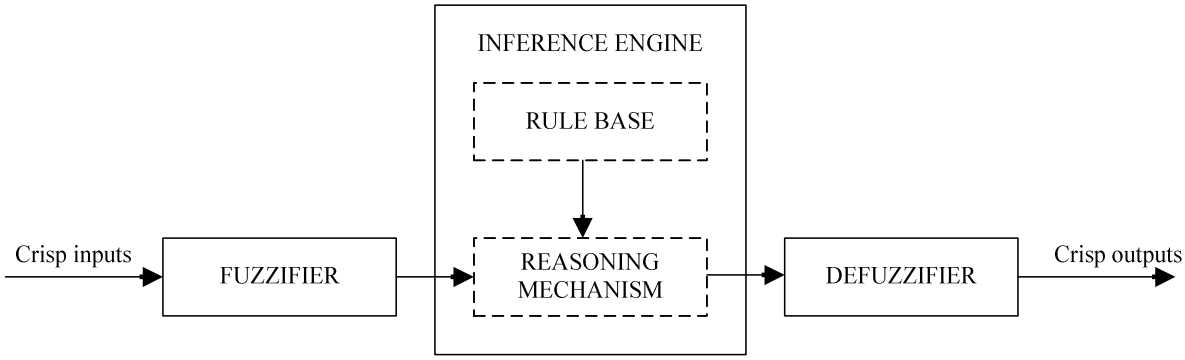


Figure 1: Fuzzy Inference System

Fuzzifier

Fuzzifiers transform crisp input values to membership grades in the range of 0 and 1 through the use of membership functions. It is worth noting that an input can have partial memberships in more than one membership function, thus resulting in more than one membership value. For example, the fuzzy variable speed in the universe of discourse X = [0, 220] can be described by three linguistic terms: low, medium and high, as shown in Figure 2. An input of 50 km/h in this case has partial memberships of 0.3 and 0.7 in the membership functions of low and medium, respectively. In other words, it is not fully a low speed, but not fully a medium speed either.

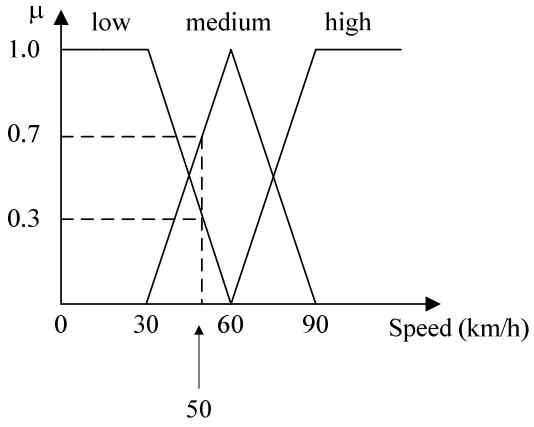


Figure 2: Fuzzification on Vehicle Speed

Inference Engine

There are two essential components in the inference engine: fuzzy rule base and fuzzy reasoning mechanism. The rule base consists of a set of if-then rules that incorporate human expert knowledge

for a particular process. These rules include fuzzy linguistic terms commonly used by human experts. The number of rules in the rule base is highly dependent on the complexity of the problem at hand. More rules are required to derive a conclusion for a problem with higher complexity. A set of fuzzy rules is essentially a collection of conditional statements in the form of:

IF x is A THEN y is B

where x and y are linguistic variables while A and B are linguistic values defined by the fuzzy sets in the universe of discourse X and Y, respectively. For example, x and y can take on the linguistic variables obstacle and deceleration while A and B can be linguistic values near and high. The IF-part and the THEN-part of fuzzy rules are generally termed as the antecedents and the consequents, respectively. Both the antecedent and the consequent of a fuzzy rule can have multiple parts. For example, the following rule has two antecedent and two consequent components:

IF obstacle is *near* AND speed is *fast* THEN deceleration is *high* and left steering is *large*

Interpretation of the antecedent in a fuzzy rule results in a numerical value in the range of 0 to 1, representing the degree of truthfulness of the statement. For rules with multiple parts in their antecedent, fuzzy operators are utilized to resolve the statement into a single number. In the interpretation of fuzzy rules, the antecedents are first evaluated and the corresponding results are then applied to the consequents.

The second part of the inference engine is the fuzzy reasoning mechanism that models the human decision-making process. This component achieves fuzzy reasoning by applying Generalized Modus Ponens (GMP) to the rule base. GMP states that given the fact "x is A'" and the rule "if x is A then y is B", it can be concluded that "y is B'" [44]. In other words, given system inputs representing the facts in combination with the rule-base consisting of fuzzy rules, conclusions can be drawn. Obtaining a conclusion from the system inputs and the rule-base involve two steps: rule evaluation and rule output aggregation. In the rule evaluation phase, decisions from individual rule in the rule-base are computed. These rule outputs are then combined to generate the final conclusion in the rule output aggregation phase. To obtain a conclusion from a single rule in the rule base, evaluation of the antecedent is necessary. For an antecedent with a single part, the evaluation result is represented by the fuzzified system input. For antecedents with more parts, fuzzy operations are applied to the membership values corresponding to the system inputs. Assessment of an antecedent provides a single numerical value, representing the level of truthfulness of the antecedent. This number, commonly termed as a weighting parameter, is utilized to adjust the membership function of the corresponding rule consequent for generation of the rule conclusion. A consequent membership function can either be truncated or scaled by the weighting parameter. The next step in generating the overall conclusion is to combine individual rule outputs. In this step, all rule consequents are unified into a single fuzzy set through the fuzzy OR operator.

Inference Models

There are various inference models. The two most commonly used models are of the Mamdani [45] and the Sugeno [46] type. In the first approach, consequent membership functions are fuzzy sets. The inference process results in a fuzzy set that requires defuzzification (a processing step required for obtaining a crisp output value). Unlike the Mamdani model, consequent membership functions in the Sugeno model are fuzzy singletons. These functions have membership value of 1 at a single point on the universe of discourse and a value of zero for all other locations. Consequents in the Sugeno model are a function of its rule antecedents. They are commonly represented by first-order or zeroth-order equations due to simplicity and efficiency. In zeroth-order Sugeno models, the consequent membership functions are basically constant values. An example of a fuzzy rule in such inference system is as follows:

IF distance is *short* AND speed is *fast* THEN deceleration is *3*speed - distance*

The rule output aggregation step in a zeroth-order Sugeno inference process combines truncated or scaled fuzzy singletons. To generate a crisp overall output, the weighted average of all rule conclusions is computed according to the following equation.

$$\hat{c} = \frac{\sum_{i=1}^{n} \omega_i c_i}{\sum_{i=1}^{n} \omega_i} \qquad (1)$$

where n specifies the number of fuzzy rules, c_i represents the consequent of rule i, ω_i is the weighting parameter for rule i and $\hat{c}$ is the overall crisp output.

Defuzzifier

Defuzzifiers map output values from fuzzy sets back to crisp numerical values. In the Mamdani approach, a defuzzifier is required. However, in the Sugeno Fuzzy Model, since the overall crisp output is determined as a weighted average of the individual rule inference, a defuzzification mechanism is not required.

3.1.2 Neural Networks

The human brain is essentially a highly efficient and effective information processing system, with the ability to make many decisions within a short timeframe. It also possesses the ability to interpret uncertain information. Biologically, the brain is a massive network of nerve cells, with a large number of interconnections among them. These interconnections provide humans with the ability to learn and to differentiate right and wrong through experience. Inspired by characteristics of the human brain, researchers developed artificial neural networks to resemble reasoning processes in these biological units. With neurons serving as basic building blocks and large number of parallel interconnections linking these processing elements, artificial neural networks are capable of learning.

Through the learning process, neural networks can acquire not only the ability to approximate nonlinear functions, but also the capability to extract hidden relationships among input and output patterns. Similar to fuzzy systems, neural systems do not represent system dynamics as a set of complicated mathematical equations, but extracts hidden relationships between inputs and outputs from a given set of training data. The effectiveness and suitability of neural networks are highly dependent on the availability of good training data that are free of noise and well distributed over the spectrum of operating conditions.

Processing Units - Neurons

Neural networks are constructed from a large number of basic processing units called neurons. They are connected together via weighted links. By proper adjustments of the associated weights, a neural network can realize diverse functionalities. An individual neuron has a number of input links, a processing node and a single output channel for signal reception, computation and transmission. Its output is calculated by applying the weighted sum of its inputs to a nonlinear mapping function called activation function. In mathematical sense, it is characterized by the equation

$$y = f\left[\sum_{i=1}^{n} \omega_i x_i + \theta\right] \tag{2}$$

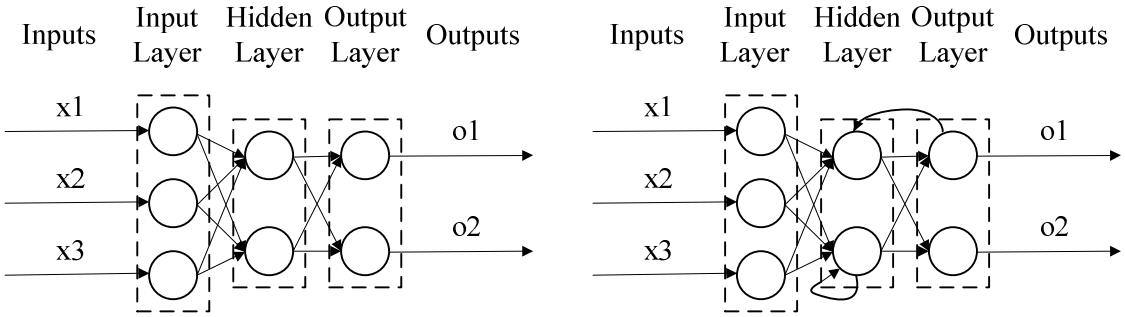
where y represents the neuron output, $f[.]$ is the activation function, x_i is the i-th input signal to the neuron, ω_i denotes the modifiable connection weight from x_i to the neuron, and θ is a modifiable bias or threshold value. Activation functions utilized in neural networks are not unique. They can be linear or nonlinear. The most common choices are linear, saturating linear, sigmoid, hyperbolic tangent and radial basis functions [43].

Network Architecture

A single neuron is inadequate in detecting patterns or providing generalization. It is the interactions among neurons that give rise to the computing abilities of neural networks. Neurons in neural networks are generally organized into layers. Each network consists of an input layer, an output layer and one or more internal layers called hidden layers. They have different roles for providing various functionalities. The input layer is free of computational units. It simply accepts input signals and re-distributes them to neurons in the following layer. On the other hand, the output layer is designed to provide the network output signal, given outputs of its previous layer. Most traits of neural networks evolve from the arrangements and bonds of neurons in the hidden layers. They provide the computational and learning capabilities. Features in training datasets are often extracted in these layers.

Neural networks can be classified by the direction of information flow. In a feed-forward neural network, neurons in different layers are connected with uni-directional (forward) paths. In other words, neuron outputs of one layer are connected solely to neuron inputs in the preceding layer, as

shown in Figure 3 (a). Aside from the feed-forward architecture, neurons can also be arranged in a recurrent manner. Recurrent networks allow bi-directional (forward and backward) connections among layers. Output connections from one layer are not limited to the inputs of the following layer; feedback connections are also permitted, as demonstrated in Figure 3 (b). Feed-forward neural networks are commonly employed due to their simplicity and their potent capability in representing input-output relationships. However, recurrent networks are also widely applied due to their information storage capabilities and dependencies on previous outputs or states.



(a) Feed-forward (b) Recurrent

Figure 3: Two Typical Neural Networks

Learning Algorithms

The main feature of neural networks is their ability to learn. In the learning process, weights connecting various neurons are constantly adjusted until the desired network output values are produced. There are three main mechanisms for learning: supervised, unsupervised and reinforced. Supervised learning refers to learning acquired through practice. The system in training is presented with a set of input-output pairs. During training, the calculated network outputs are compared with the desired output values to formulate an error signal. This signal is then utilized in the training algorithm to update the weights with the ultimate goal of minimizing the differences between the actual and the desired network outputs. Frequently used algorithms for supervised learning include Back-Propagation Learning (BPL) [47] and Levenberg-Marquardt Learning [48]. Unsupervised learning involves learning through well-defined guidelines [43]. The network attempts to cluster input patterns and extract hidden features from the training set. In this learning mechanism, neurons in the output layer are evaluated in performance and compared with each other based on set rules. Through these competitions, weights are updated to strengthen connections between input neurons and the winning output neuron. Often, connections to neighboring output neurons are also emphasized. Through competitions in unsupervised learning, hidden relationships within training sets are discovered. Kohonen self-organizing network [49] is an example of this type of learning. In reinforcement learning, the network acquires the appropriate weights through a credit and punishment system. Rewards and penalties are generated to strengthen or weaken a particular connection from

the input layer to the output layer based on the correctness of the network response. Among all reinforcement learning schemes, reinforcement comparison methodology and Q-learning algorithm [50] are of great interest.

3.2 Reconfigurable Hardware

Conventionally, most engineering designs are realized either on standard components such as General-Purpose Processors (GPPs) or through application-specific parts such as Application Specific Integrated Circuits (ASICs). For applications using GPPs, implementations are performed in software due to flexibility and short development time. Design changes can be attained timely by modifying software programs. Short development times are obtained because hardware configurations are unnecessary. However, these good qualities come at the expense of performance. Requirements for instruction interpretation and sequential execution limit processing speeds of GPPs. In Application Specific Integrated Circuits (ASICs), hardware is customized for a particular application. Logic units are hard-wired onto silicon chips in fabrication processes, providing efficient use of silicon area. In hardware design, independent control units can be executed in a parallel and concurrent manner, giving superior performance. Since logic design and wiring are customized and fixed at fabrication time, ASIC design is time-consuming and costly. Any changes will lead to additional delays and costs. For large volume production, the use of ASICs may be justifiable. However, for low volume production or prototyping purposes, the inflexibility and lengthy development time of ASICs are unacceptable.

In general, cost and speed are the most prevalent parameters in system designs. Software designs feature flexibility and short development time at the expense of performance. On the other hand, hardware designs lack flexibility but provide superior performance. High performance systems are desired, but flexibility and minimal development time are also important as they can be translated directly to lower production costs. The development of Field-Programmable Gate Arrays (FPGAs) provides the benefits of both platforms. Utilizing standard components that are readily programmable, FPGAs are an alternative to GPPs and ASICs. Through the use of reconfigurable hardware elements, the FPGA platform offers the benefits of GPPs in terms of programmability and adaptability along with the advantages of ASICs in terms of speed and compactness. For large volume production, ASICs will always be more advantageous compared to FPGAs in terms of reduced chip area and increased operational speed due to the absence of programmable interconnect circuitries and delays. Thus, it is a common approach for designers to build prototype systems on FPGAs during the development phase and later migrate to ASICs for production. Aside from prototyping, an emerging area called reconfigurable or run-time reconfigurable computing makes use of the reconfigurable capability of FPGAs. In these applications, components of an FPGA are reconfigured during run-time to realize different functionalities at different time steps.

3.2.1 FPGA Architecture

A FPGA is essentially an array of functional blocks with a network of interconnects, which can be reconfigured at the time of use. Different functionalities can be achieved through reconfiguring individual blocks and changing the interconnections among them. The smallest logic unit in a FPGA is a Logic Element (LE). A FPGA usually contains a large number of these programmable units. For example, the Stratix EP1S FPGA [51] from Altera Corp. contains 79040 LEs. A typical LE consists of a four-input Look-Up Table (LUT), a programmable register, a cascade chain and a carry chain, as illustrated in Figure 4. The LUT is essentially a 16 by 1 memory unit, which has the ability to realize any function with four variables. It is preloaded with the output values for all combinations of inputs of the function to be implemented. The appropriate output value can be selected by utilizing the four input signals of the LUT as the address. The LUT is generally utilized for implementation of combinatorial logic, providing predictable delays and area efficiency [52]. A single LE is generally insufficient for implementation of most hardware designs. Thus, the output of the LUT is usually first fed into a programmable register, then connected to other LEs. Depending on the application at hand, the register can be configured as a D flip-flop, a T flip-flop, a JK flip flop or a SR latch. In addition to the flip-flop, the carry and the cascade chains are also designed to facilitate computations involving more than one LE. The carry chain provides an efficient carry-forward function to the adjacent LEs, allowing implementation of high-speed counters and adders [53]. The cascade chain, on the other hand, is utilized in logical operation requiring more than four inputs provided by a single LUT or LE. In this situation, adjacent LEs are utilized to compute portions of the function and the cascade chain is employed for connections of the intermediate values.

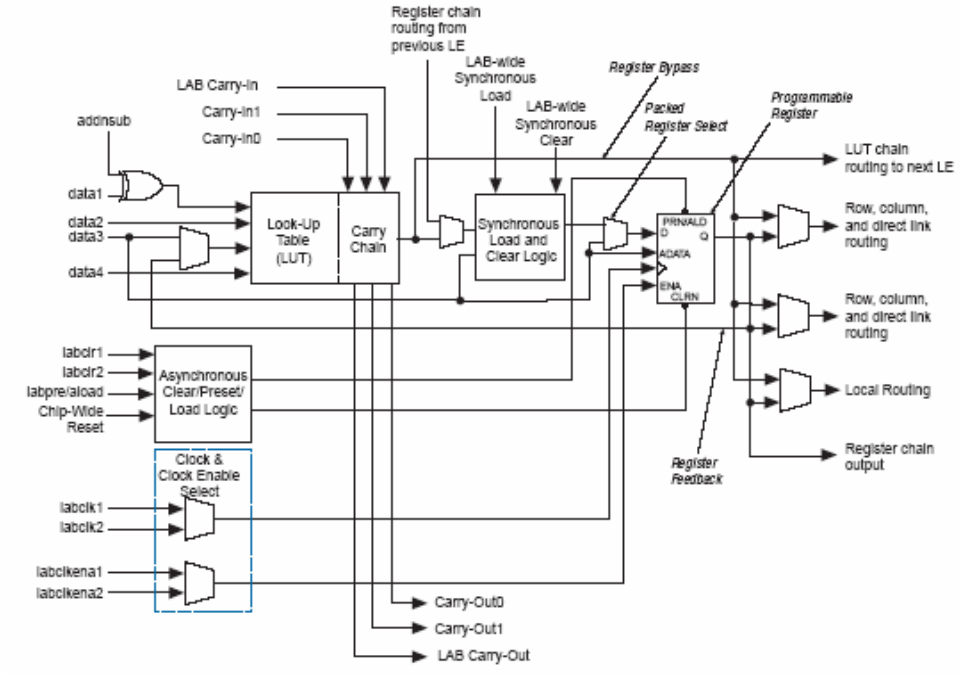


Figure 4: Internal Structure of a Logic Element [51]

Complex functions or logic operations generally require the use of more than one LE. For high performance and efficient use of silicon area, LEs are clustered into LABs. Depending on the FPGA selected, the number of LEs in a single LAB varies. For example, a LAB in the Stratix EP1S FPGA contains 10 LEs while that of the Flex 10K FPGA has 8 LEs. LABs are arranged in a matrix form and communicate to other LABs through interconnects (or wires). These connections can be categorized into two types: FastTrack and Local FastTrack [54]. The first spans the entire chip either horizontally or vertically, providing low-delay paths for communications among different LABs. They also carry signals to and from the device's Input/Output pins. The number of FastTrack in a FPGA is generally limited as they are relatively expensive in terms of device area. Instead of spanning the entire chip, Local FastTracks are limited only to LEs in the same LAB. They provide high-speed communication for inter-LAB LEs. Aside from LABs, FPGAs also provide special purpose blocks dedicated for memory storage (RAM), arithmetic multiplication and addition operations (DSP), and communications with other peripheral devices (IOE). These blocks are often optimized for specific tasks. Figure 5 illustrates the architecture of a FPGA in the Altera Stratix family, with indications of the location of LABs, RAM blocks, DSP blocks and IOEs.

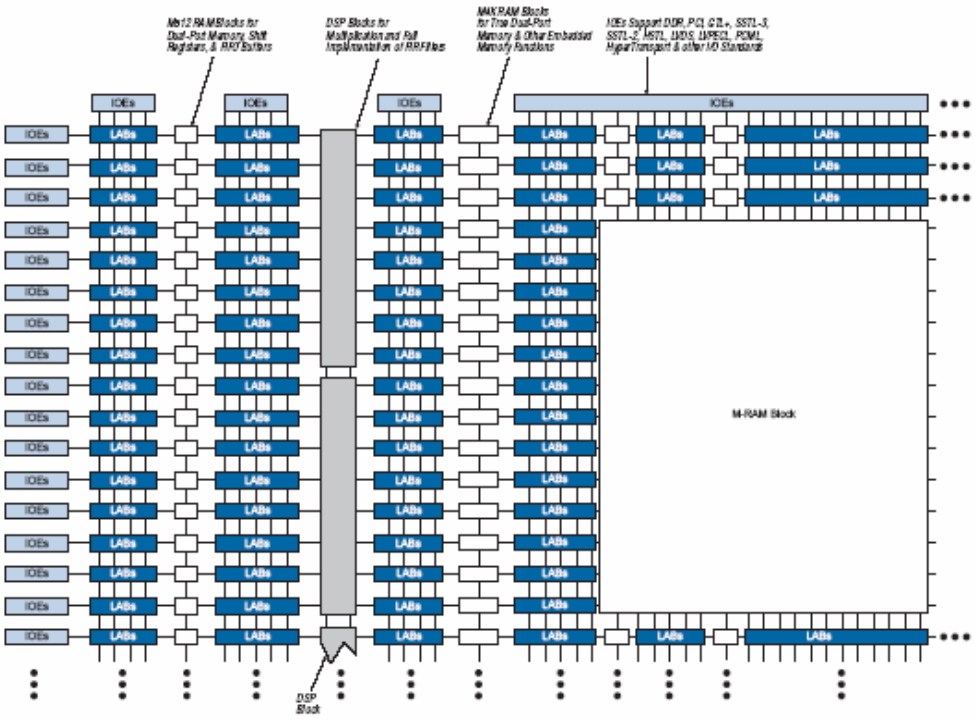


Figure 5: Stratix FPGA [51]

3.2.2 FPGA Design Flow

FPGA design flow starts with the design entry stage and ends with programming onto the target hardware unit. It involves five main steps and two optional steps (shown in solid and dotted blocks, respectively), as illustrated in Figure 6. The first step in the design flow is to specify the design. Traditionally, this is performed by drawing each hardware component in a design tool to form schematic diagrams. These diagrams represent hardware components and interconnects as blocks and line segments, making connections among hardware units easy to comprehend. However, with systems of increasing size and complexity, this approach is becoming too time-consuming and inefficient. Similar to the association of high-level programming language C to software design, Hardware Description Languages (HDLs) are widely adopted as a means for description of hardware designs. The most commonly used HDLs are Very High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog HDL. These languages provide high portability, allowing identical design interpretation from different design tools. They also reduce the overall development time and costs. Performance of a specific design in different hardware technologies can be easily evaluated. HDLs differ from conventional programming languages, in which they realize concurrent statements instead of sequential statements; since digital hardware is parallel in nature, the use of sequential logic is inappropriate. For systems requiring the integration of a large number of complex components, a combination of HDL coding and schematic diagrams is often used. Note that most development tools have the ability to translate schematic diagrams into HDL coding during compilation to avoid compatibility issues.

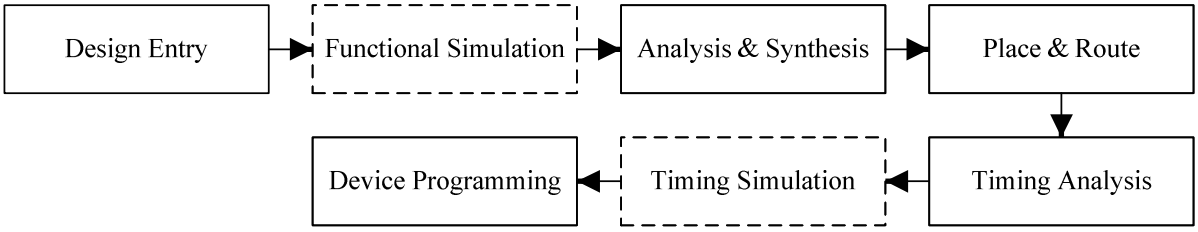


Figure 6: FPGA Design Flow

Once the design is created, it is ready for compilation by the selected Computer Aided Design (CAD) tool. The compilation process generally involves the following steps: analysis and synthesis, place and route, and timing analysis. However, this may differ somewhat from one CAD tool to another. In the analysis and synthesis stage, the design is first analyzed such that all descriptions, either in HDLs or in schematic diagrams, are translated into a uniform format. The translated design is then turned into a generic hardware format, such as a set of Boolean expressions or a netlist, early in the synthesis stage [54]. A netlist is simply a list that marks the gate-level representation of the design, illustrating basic components such as logic gates and flip flops. The next step is to optimize the design given a set of Boolean expressions or a netlist. The goal of this step is to remove redundant logic expressions and to minimize the number of logic gates required. With an optimized

design, the CAD tool then matches the logic gates to LEs of the FPGA. The place and route phase maps different components and interconnections into specific locations on the FPGA. This step is usually technology dependent with different routings and placements according to the layout of the FPGA. The output of this step is a bit stream file, which can be downloaded onto the target device. Prior to programming the target FPGA device, satisfaction of the performance constraints is verified in the timing analysis stage. The critical path, defined as the slowest path on the FPGA between two flip flops or a flip flop and an input-output pin, is calculated to determine the maximum clock speed. Other timing related factors, such as worst-case delays, setup and hold times, are also computed to ensure various timing requirements are met. The last step in the design flow involves programming the target FPGA device. In this stage, internal structure of the FPGA is reconfigured. LEs along with their corresponding interconnects are configured for realization of different functionalities.

Design verification is an important part of digital design. Most CAD tools provide users with two methodologies to simulate their designs: functional simulation and timing simulation. Functional simulation is generally performed following the design entry step to eliminate syntax errors and to ensure the correctness of logic flow. During this simulation process, a test-bench consisting of a set of stimuli is tested on the design. The simulated outputs are then compared to the desired output for detection of logical errors. Functional simulation is designed to verify behavior of a hardware design, with no considerations on gate and wire delays. Unlike functional simulations, timing simulations account for propagation delays of logic components and their associated interconnects. It is generally performed prior to target device programming to ensure correct responses will result when the compiled design is mapped on the FPGA unit. By applying test patterns to the inputs and comparing the simulated outputs to the desired results, any timing violation becomes evident. As timing simulation makes use of the routed and placed design, its results are dependent on the target device. Note that due to the additional timing considerations, it is possible to have the functional simulation behavior differ from that of the timing simulation. If this occurs, design modifications are required.

3.2.3 Intellectual Property (IP) Cores

To shorten development time, many FPGA design tools provide users with a set of pre-designed hardware components, termed intellectual property (IP) cores. IP cores are generally complex in design, but serve a wide variety of applications [55]. For example, NIOS II [56] and FFT [57] are two IP cores offered by Altera Corp. for realization of an embedded microprocessor and a digital signal processor, respectively. Ideally, IP cores should be portable for use in different design tools. However, they are often optimized for performance and cost for only a limited number of FPGAs. IP cores are generally available in one of the following three forms: hard, firm and soft. Hard cores are optimized in performance, area and size. They are mapped to a specific process technology and provided to users in a fully placed and routed format [55]. In terms of flexibility and portability, hard cores are inferior to firm and soft cores as their layouts are fixed. Similar to hard cores, firm cores are also optimized in performance and area. However, they offer the flexibility of being reconfigurable to various applications as they are often presented in the form of synthesized code or a netlist of

generic library elements. Soft cores are the most flexible form of all three. They are either described by HDL code or exist as a netlist of logic gates and associated interconnects. Functionalities and implementations of soft cores can be modified easily according to the application at hand. In this thesis, a soft microprocessor core, NIOS II, is utilized for validating the hardware based fuzzy logic and neural network controllers.

3.3 Vehicular Model

Dynamics of vehicular systems are highly complex and nonlinear in nature. According to Alonso [58], one of the most commonly used vehicle model assumes fourteen degrees of freedom, with six allocated for translation and rotation of suspended mass, four for vertical movement of non-suspended mass and the remaining four accounted for dynamics in wheels rotation. Some common parameters in vehicular control includes the front and rear longitudinal forces, the linear and angular velocities and the position of the vehicle's center of mass with respect to the front and rear axles. Many researches are based on a simplified vehicular model, governed by the following set of equations:

$$\dot{x}_f = v\cos(\theta + \phi) \tag{3}$$

$$\dot{y}_f = v\sin(\theta + \phi) \tag{4}$$

$$\dot{x}_r = v\cos\theta\cos\phi \tag{5}$$

$$\dot{y}_r = v\sin\theta\cos\phi \tag{6}$$

$$\dot{\theta} = \frac{v\sin\phi}{l} \tag{7}$$

where x_r and y_r represent the position of the rear wheels, x_f and y_f represent the position of the front wheels, θ represents the angle between the x-axis and the car wheel base line, v represents the velocity of the car, and ϕ represents the steering angle of the car. However, this model is often over-simplified as it has no considerations on surface friction, tire conditions and wear and tear in other mechanical components. Also, each pair of wheels is modeled as a single wheel located at the midpoint of both front and back axles. Although the above is a simplified model, the high nonlinearity and complexity in vehicle dynamics is still apparent. As an attempt to illustrate the high level of coupling dynamics in actual vehicles, the Longitudinal Vehicle Dynamics Model [59] by Mathworks Inc. is presented. In this model, the forward and backward motion of a vehicle, with two-axle and four equally sized wheels, along its longitudinal axis is modeled with the following set of equations:

$$m\dot{V}_x = F_x + F_d - mg\sin\beta \tag{8}$$

$$F_x = F_{xf} + F_{xr} \tag{9}$$

$$F_d = -\frac{1}{2} C_d \rho A V_x{}^2 \operatorname{sgn}(V_x) \tag{10}$$

$$F_{zf} = \frac{h(F_d - mg\sin\beta - m\dot{V}_x) + bmg\cos\beta}{a+b} \tag{11}$$

$$F_{zr} = \frac{-h(F_d - mg\sin\beta - m\dot{V}_x) + amg\cos\beta}{a+b} \tag{12}$$

where g represents the gravitational acceleration of -9.81 m/s^2, m is the mass of the vehicle, A is the effective frontal vehicle cross-sectional area, h is the height of the vehicle's center of gravity above the ground, a and b are the distances of the front and the rear axles from the vertical projection point of the vehicle's center of gravity onto the axle-ground plane, C_d is the aerodynamic drag coefficient, ρ is the mass density of air of 1.2 kg/m^3, $|F_d|$ is the aerodynamic drag force, F_{xf} and F_{xr} are the longitudinal forces on the vehicle at the front and rear wheel ground contact points, β is the inclination of the vehicle, F_{zf} and F_{zr} are the vertical load forces on the vehicle at the front and rear ground contact points and V_x is the longitudinal vehicle velocity. In this model, both friction and wind resistance are accounted for, showing that changes in these variables have an effect on the longitudinal velocity of the vehicle, effectively modifying the overall velocity of the vehicle in the previous model. It is important to note from this model that the dynamics of vehicular systems are highly coupled and nonlinear. A single variable, such as the longitudinal velocity, is dependent on a number of different parameters. This makes soft computing control methodologies more attractive in vehicular control compared to their classical counterparts.

3.4 Summary

Classical control techniques require the use of analytical models. The development of an effective classical controller for industrial processes with nonlinear characteristics often poses a challenge. In addition, the incorporation of human expertise and experience into these systems is difficult to achieve. This chapter presents the basic phenomenon of two widely applied soft computing techniques, fuzzy logic and neural networks. They are based on human reasoning and learning paradigms, eliminating the need for complex mathematical models. To demonstrate the suitability of soft computing models in vehicular control problem, this chapter provides a brief discussion of conventional vehicular dynamics and modeling. The high level of nonlinearity and coupling among different parameters are demonstrated. For design of a vehicular prototyping system, this chapter also provides an introduction on reconfigurable hardware. The internal structure of a FPGA is given along with the steps involved with implementing a design onto a FPGA.

Chapter 4
Experimental Setup

To facilitate research, a prototype vehicle with two different modes of operation is developed. It can either be controlled by a human operator through a customized control pad or navigated autonomously based on the selected intelligent controller. In brief, the prototype vehicle is made up of the following components: a main chassis, a sensory system, a control pad, a control unit, a locomotion mechanism and a steering control mechanism. These components provide different functionalities. For example, the sensory system percepts and gathers environmental information, the control unit hosts the reasoning mechanism for generation of motion and steering commands, and the locomotion and steering mechanism execute the control instructions. This chapter focuses on the development of the prototype vehicle. Hardware design considerations are provided.

4.1 Prototype Architecture

The prototype vehicle is built upon a commercial radio control racing car package, Subaru Impreza WRC 2004, manufactured by Tamiya Inc. The physical dimensions of the chassis are 441 mm x 185 mm x 137 mm (Length x Width x Height), approximately one-tenth of that of an actual vehicle. Aside from the main chassis with wheels and tires, the commercial car package also includes the following parts: bumpers, suspensions, a gearbox, a speed controller, a DC motor and a steering unit. Most hardware components utilized in the prototype system is included in the package, with the exception of the sensory system, the control pad and the FPGA-based control unit. Figure 7 presents the overall architecture of the prototype system. The control pad and the sensory system provide inputs to the control unit, which in turn generates steering and locomotion commands for control of the vehicular system.

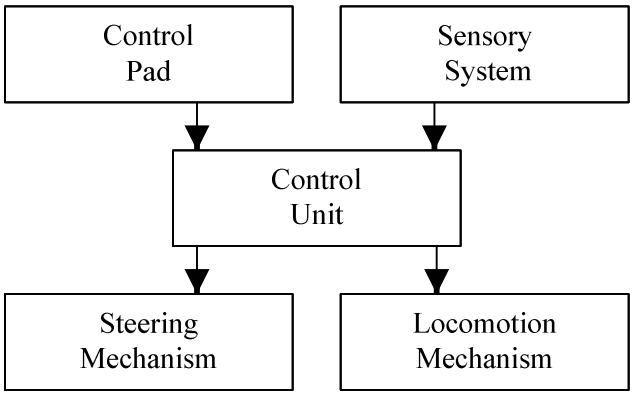


Figure 7: Overall Architecture of the Prototype Vehicle

4.2 Sensory System

Perception is a prevalent component in the design of vehicular systems. Human operators base their driving decisions on their vision while autonomous vehicular systems depend on their sensing units. To obtain information of the surrounding environment, autonomous systems are often mounted with various types of exteroceptive sensors. The most common type of these sensors is range sensors, which have the ability to measure the relative distances between the vehicle and the objects in its vicinity. Three types of range sensors are considered in this research work. They are the sonar, the laser and the infrared sensors. Sonar sensors, characterized by their low costs, were widely adopted in the past. In sonar sensors, sound waves are transmitted and the time taken for the wave packet to reflect and return to the receiver is measured. They are based on the time-of-flight principle in which the relationship d = ct is used to estimate the distance measurement, where d represents the distance traveled by the wave or two times the distance between the vehicle and the object, c is the constant speed of the wave propagation and t is the time-of-flight measurement. Since sound wave measurements are easily affected by interferences and reflections, the uses of sonar sensors have decreased in popularity. Laser rangefinders, on the other hand, have attracted much attention in recent years. Also based on the time-of-flight principle, laser rangefinders utilize laser light instead of sound waves to acquire range measurements. Due to the high traveling speed of light and the coherency of laser beams, this type of rangefinder is characterized by high bandwidth and high accuracy. The main drawbacks in these sensors lie in their heavy weights and high costs. They are generally unsuitable for small mobile applications. For low budget range measurements, infrared sensors are highly valued. Instead of measuring the time, the operations of infrared sensors are based on geometric properties. Known light patterns are projected to the environment and the corresponding reflections are captured for geometric calculations. Resolutions of infrared sensors are inversely proportional to distance. In addition, their operating ranges are limited by geometry. Nevertheless, they usually provide a wide enough range of distances for common range-finding applications.

4.2.1 Infrared Sensors

Considering issues such as cost and bandwidth requirements, infrared sensors are deemed as the most appropriate sensor type for the current application. Three widely used infrared sensors are GP2D120 [60], GP2D12 [61], and GP2Y0A02YK manufactured by Sharp Corp. They differ in detection range, as shown in Table 1. Together, they cover most detection distances required in small scale vehicular prototyping systems. For variable range measurements, the prototype is equipped with four short-range GP2D120 and two mid-range GP2D12 sensors. Their corresponding positions are presented in Figure 8, where DF and DR represent the range measurements from the front and the rear short-range sensors, DSFM and DSBM are the distance measurements from the side front and the side back mid-range sensors while DSFS and DSBS correspond to the relative distances from the side front and the side back short-range sensors to the surrounding environment. The front and rear sensing units are

equipped mainly for obstacle avoidance purpose. In unforeseen circumstances, in which an obstacle is in the path of the vehicle, the front and back sensors will prevent the vehicle from crashing, while traveling either in the forward or the reverse mode. The side sensors, which are mainly for wall-following purpose, are placed relatively close to the front and the back on one side of the vehicle to ensure that both the orientation of the car with respect to the wall and the relative distance between the vehicle and the wall can be accurately determined. Note that if the side sensors are placed too close together, information near the front and the rear of the car would be outside of the sensors' detection zones and will not be taken into account.

Table 1: Detection Range for Three Infrared Range Sensors

Model	Detection Range (cm)
GP2D120	4 – 30
GP2D12	10 – 80
GP2Y0A02YK	20 – 150

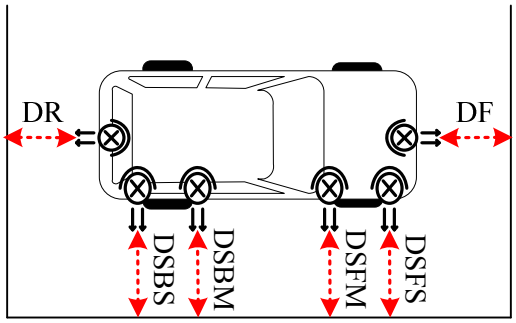
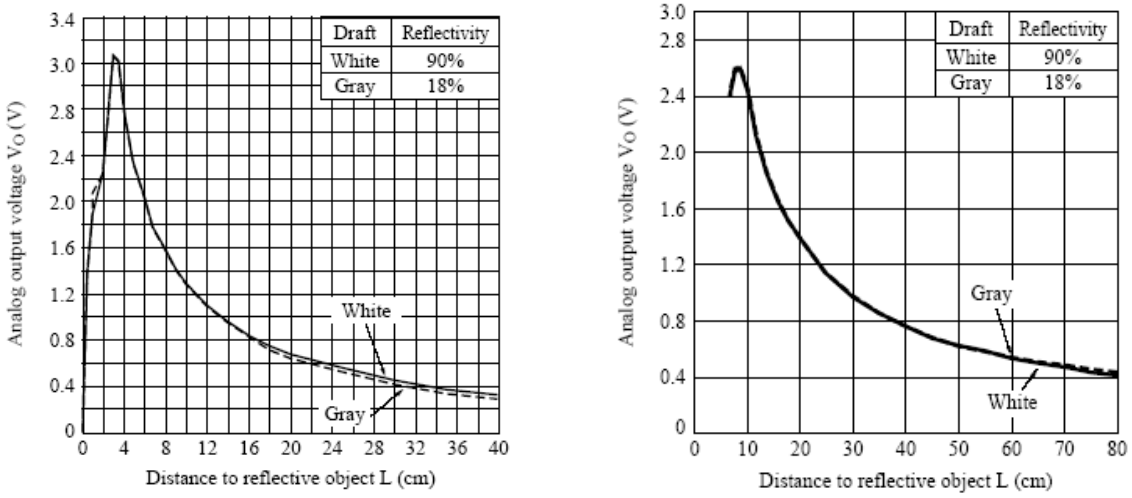


Figure 8: Sensors Configuration

The range sensors equipped on the prototype represent detected distances as analog voltages. These voltage-distance relationships are nonlinear in nature, and vary from one sensor to another. Figure 9 illustrates two voltage-distance characteristic curves for GP2D120 and GP2D12, respectively. It can be seen that when the sensors are utilized within the reliable detection ranges, the output signals are inversely proportional to the distance measurements. Also, as an example of how much variance the output voltage value of two different sensors can be, a distance of 20 cm is represented as a voltage of approximately +0.7V and +1.4V by GP2D120 and GP2D12, respectively. Since outputs of the range sensors are of the analog type, an Analog-to-Digital Converter (ADC)

component is necessary to convert sensory information into digital format for further processing by the main control unit realized on a FPGA board.



(a) GP2D120 (b) GP2D12

Figure 9: Infrared Sensor Voltage-Distance Mappings [60]-[61]

4.2.2 Analog-to-Digital Converter

To digitize the sensory signals, a 10-bit resolution high-speed parallel output ADC, ADC10461 from National Semiconductor, is chosen after considering its benefits in resolution, bandwidth and cost. At an operating voltage of 5V, ADC10461 transforms analog input signals into 10-bit digital output signals. Figure 10 presents the schematic design of the selected ADC. V_{IN} represents the analog input channel of the ADC while DB0 to DB9 represent the digital output channels. /S/H represents the sample/hold control input, controlled by the main control unit. The analog input signal is sampled and the conversion process is initiated when this pin is set to the low state. /RD represents the active low read control input, also controlled by the main control unit. The output channel assumes the converted digital signal when this pin is set low. /INT is the active low interrupt output. It assumes a low value at the end of each conversion and returns to a high state following the rising edge of /RD. For other pin configurations, refer to the datasheet of ADC10461 [62]. Ideally, the output pins of the sensors are connected to the input channel of the ADC. The ADC then quantifies the sensor values and outputs the corresponding digitized sensory data to the control unit. However, due to the limited number of input channels in the selected ADC, V_{IN} is connected to the output of a multiplexer unit instead. In this case, the six sensor outputs are connected to the input channels of the multiplexer unit.

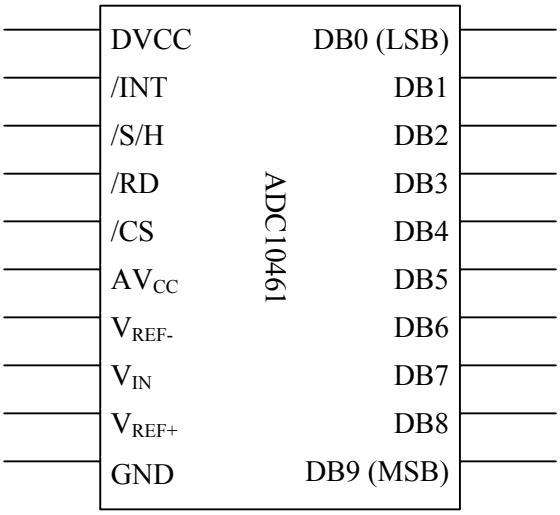


Figure 10: ADC10461 Schematic Diagram

4.2.3 Multiplexer

A multiplexer contains several input channels and a single output channel. It allows the selection of information from a particular input channel for the output channel. The selection process depends upon the values of the control pins. For example, in a multiplexer that allows for the selection of the front sensor output or the rear sensor output, both the front and the rear sensor signals are connected to the multiplexer inputs. The control bit is assigned a value of 1 for the front sensor and 0 for the rear sensor such that when one of the two possible values is chosen, the corresponding output is chosen for the multiplexer. With six sensory outputs but only one ADC input channel, multiplexer circuitries are required to successively multiplex data from each sensor to the input of the ADC. An 8-channel analog multiplexer, MC14051B, is chosen to perform the task. Figure 11 illustrates the schematic design of the multiplexer. X0 to X7 represent the eight analog input channels and X carries the output signal. Pins A, B and C are utilized for input channel selection, with the available choices summarized in Table 2. For other pin configurations, refer to the datasheet of MC14051B [63]. With this multiplexing unit, the six sensory data can be digitized one at a time by the ADC. To multiplex the input channels successively, signals A to C are controlled directly by the main control unit of the system.

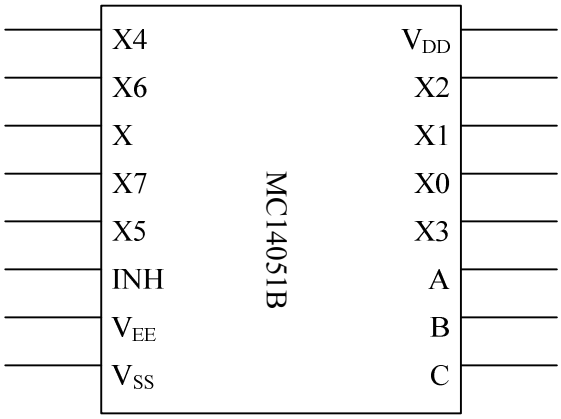


Figure 11: MC14051B Schematic Diagram

Table 2: Truth Table for Multiplexer MC14051B

C	B	A	Selected Output
0	0	0	X0
0	0	1	X1
0	1	0	X2
0	1	1	X3
1	0	0	X4
1	0	1	X5
1	1	0	X6
1	1	1	X7

4.3 Control System

4.3.1 Control Pad

To facilitate in the manual control mode of operation, the prototype system is equipped with a customized control pad. It is composed of several push-button switches and resistors. The switches are of the normally open type, in which the two mechanical contacts on the switches touch to form a circuit when pressed and separate to break the circuit when the button is released. Steering commands can be inputted through action buttons on the control pad, which outputs directly to the main control unit. The control pad contains three switches, individually controlling three output channels: S, L and R. The S channel controls a status bit, allowing an initialization signal to be sent to the main control unit when pressed. The L and R buttons are utilized for manual steering control,

in which the L button corresponds to leftward steering while the R button represents steering to the right.

4.3.2 Control Unit

The main control unit is realized on the Altera Stratix EP1S10 FPGA development board. The board hosts a FPGA with 426 input-output pins, 79040 LEs, 920448 RAM bits and 6 DSP blocks [64]. It is well-suited for applications requiring low power consumption and high performance. With sufficient circuitries, the FPGA board has the ability to simultaneously interact with different peripherals and perform several tasks. Figure 12 presents the schematic diagram of the main control unit. This component serves as the "brain" of the autonomous system. Not only does it interpret coded sensory information from the ADC into corresponding steering and locomotion control signals, it also interacts and controls other components such as the multiplexer, LED display and push-button switches.

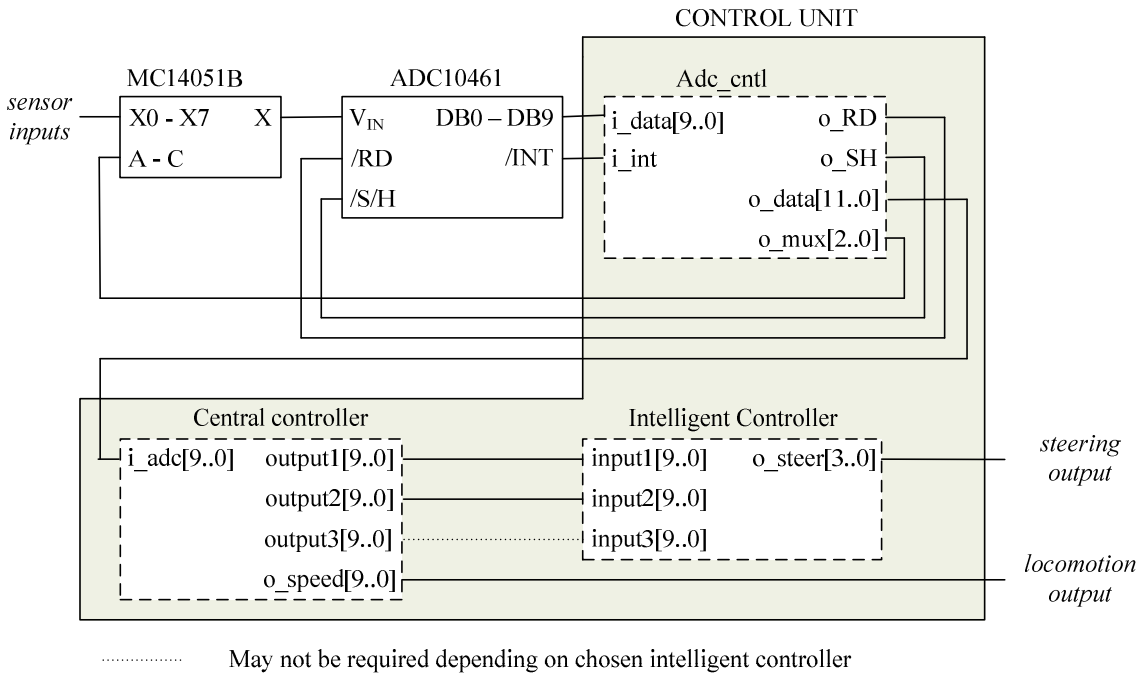


Figure 12: Schematic Diagram of the Main Control Unit

The main control unit is composed of three major components: adc_ctrl, central controller and intelligent controller. The adc_ctrl unit determines the control signals of the ADC. It manipulates the sample/hold (/S/H) and the read (/RD) signals for collection of correct digital information. It also determines the timing and sequence of the sensor selection process in the multiplexer to successively

gather sensory data from the six infrared sensors. The central controller keeps track of the internal state of the control system. It analyzes data from various sensors for obstacle avoidance purposes. At the same time, it also computes and provides the intelligent controller with the appropriate input signals. Several push-button switches are connected directly through the control pad to this controller for initiation and debugging purposes. Steering control signals are generated by the intelligent controller, which receives its inputs from the central controller. Reasoning in the intelligent controller can be based on fuzzy logic, neural network or other control paradigms. Note that to utilize different control algorithms or to perform different tasks with the same control technique, only the central controller and the intelligent controller require modification.

4.4 Actuation Mechanism

Actuation refers to the process of transforming an input signal into motion. The main goal of an actuator is to realize commands generated from the control unit. Specifically, the steering angle and the movement direction (forward or backward) are to be controlled. Commonly used actuators include standard DC motors and servos. The control concept of standard DC motors is simple. By applying a voltage on the terminals of these motors, rotations are initiated. The application of different DC power to the terminals varies the rotational speeds and the directions. In general, applying higher power to the motor units produce higher speeds, while switching the polarities of the motor terminals reverses the rotational direction. A variation of standard DC motors is servos. A servo is a small device that incorporates a DC motor, a gear train, a potentiometer, an Integrated Circuit and an output shaft. Unlike DC motors, input signals to a servo are translated to angular position instead of rotational velocity. Also, continuous rotations are not attainable in servos as they can only rotate a maximum of approximately 120^o in each direction from its neutral (0^o) position [65]. Since DC motors can revolve freely in both counterclockwise and clockwise directions with variable speeds, it is well-suited for the realization of motion control of vehicular systems. Changes in rotational velocity of the motor are translated directly to changes in wheel velocity, and correspondingly the velocity of the vehicle is altered. Servos, on the other hand, are more appropriate for the realization of steering control of vehicular systems. With limitations on their angular positions, servos closely emulate the constraints on steering angles. In driving systems, drivers can only steer a certain amount to the left or right direction from the middle position. Due to their appropriateness, a DC motor and a servo are utilized to realize the locomotion and the steering commands from the main control unit.

4.4.1 Steering Mechanism

For steering control, the servo provided in the model car package is adopted. Three wires can be seen outside of the servo case. They correspond to the power line, the ground line and the control input line. To position the shaft of the servo at a specific angular position, coded signals are sent directly to the control input line. Each coded signal represents a unique angular position. Figure 13 illustrates

three sample steering control signals for the built-in servo, which has the same characteristics as most servos with a servo period of 20 ms. To maintain the shaft at its neutral position (0^o), a coded signal with a pulse width of approximately 1.67 ms has to be sent to the servo in every servo period. Any alteration to the pulse width changes the angular position of the shaft. A pulse width less than 1.67 ms moves the servo shaft towards its leftmost position while a pulse width of more than 1.67 ms rotates the servo shaft towards its rightmost position. To closely emulate a vehicular system, the steering pulse width is limited to the range of 1.19 ms to 2.10 ms, representing steering angles in the range of -30^o to $+30^o$.

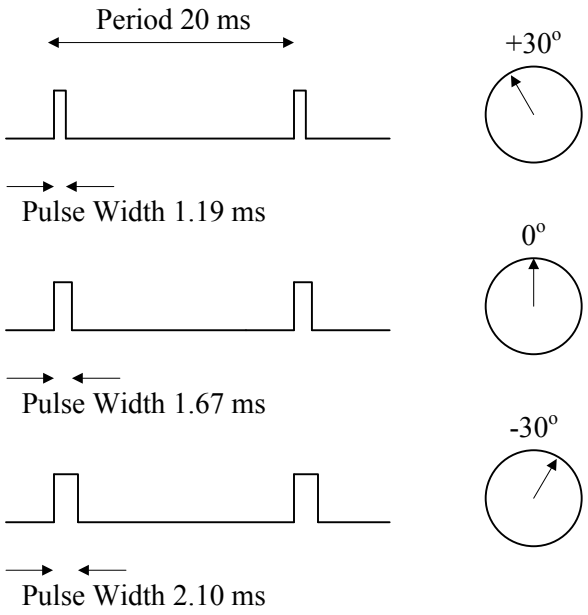


Figure 13: Three Steering Control Signals

This method of servo shaft control based on the duration of the applied pulse is called Pulse Width Modulation (PWM). PWM eliminates the need for Digital-to-Analog converters. In addition, by employing digital signal, noise effects are minimized since relatively few interference sources are able to alter a logic-1 to a logic-0 value and vice versa. Also, with digital control, power consumption can be reduced significantly. Utilizing the whole possible range of PWM signals is unnecessary and computationally demanding. Thus, for ease of implementation and to conserve computational resources, only fifteen different signals are used. Their pulse widths, steering output representation and correspondences to actual steering angles are given in Table 3. To perform a left turn of 15^o, the controller interprets from the sensory information that a 15^o left turn is necessary. It then sends a steering output value of 10 to the PWM signal generator block that resides in the FPGA board, which in turn outputs a pulse width of 1.51 ms.

Table 3: PWM Pulse Widths, Steering Outputs and Actual Steering Angles

Pulse Width (ms)	Steering Output	Steering Angle (+ = Left, - = Right)
1.1900	14	30
1.2700	13	28
1.3500	12	26
1.4300	11	24
1.5100	10	15
1.5900	9	12
1.6700	8	5
1.7100	7	0
1.7660	6	-5
1.8220	5	-12
1.8780	4	-15
1.9340	3	-24
1.9900	2	-26
2.0460	1	-28
2.1020	0	-30

4.4.2 Locomotion Mechanism

Locomotion refers to the process of causing an autonomous robot or vehicle to move [66]. The main goal of the locomotion mechanism is to realize forward and backward motions. A standard DC motor is chosen to perform these tasks due to its simplicity and availability. A DC motor consists of a positive and a negative terminal. To control the rotational speed of the motor, voltages of various levels are applied to these terminals. Similarly, to reverse the rotational direction, voltage polarity is reversed. Intuitively, the two motor terminals should be connected directly to the main control unit, which process information from sensors and output required motion commands. Unfortunately, two issues prevent such a connection. Firstly, to modify the rotational speed of the motor, analog voltages of different magnitudes are required. However, output signals of the main control unit implemented on FPGA hardware are digital in nature, providing only discrete values of 0V (off), and 3.3V or 5V (depending on the pin type). Secondly, it is difficult to reverse the polarities on the two motor terminals directly from the FPGA hardware. To overcome the two problems, speed controllers are commonly employed. Instead of establishing direct connections between the main control unit and the DC motor, digital outputs from the control unit are first connected to the input channels of a speed

controller for generation of the appropriate DC motor control signal. The output channels of the speed controller are then wired to the motor terminals.

An electronic speed controller, TEU-101BK from Tamiya Inc., is chosen due to its compatibility with the control unit. TEU-101BK generates motor control signals based on the PWM approach previously described. Instead of processing an analog voltage proportional to the rotational speed of the motor, PWM signals with pulse durations proportional to the rotational speed are utilized. A pulse width of 1.5 ms corresponds to the stop state. Any deviation to that pulse width changes the rotational velocity of the shaft. Pulse widths with higher deviations from 1.5 ms correspond to higher rotational speeds while those with smaller differences move the motor shaft slower. Also, a pulse width less than 1.5 ms moves the shaft motor counterclockwise while a pulse width of more than 1.5 ms moves the servo shaft clockwise. By employing PWM control, analog signals can be simulated from digital sources, thus allowing for effective rotational velocity control. From experimental data, it is concluded that the overall angular speed for the selected motor is relatively high. Hence, the motor is constantly set to operate at its slowest speed setting at all times.

4.5 Summary

To demonstrate the feasibility of different control algorithms, a prototype vehicle is developed. Infrared sensors are equipped on the vehicle for perception. A FPGA board is employed for controller realization. In addition, a servo and a motor are supplied for steering and locomotive purposes. The prototype vehicle is designed to support a wide range of control algorithms. As a FPGA is utilized for implementation of the main control unit, different controller designs with varying objectives can be implemented on the same prototype system by reconfiguring logics and circuitries in the control unit. The next chapter provides the design and implementation of two intelligent controllers that are tested on the prototype system.

Chapter 5
Intelligent Controllers Design and Implementation

The overall hardware architecture of the prototype is designed with considerations in flexibility. It is implemented in such a way that alterations are unnecessary in all components except the main control unit, even if different algorithms are utilized for vehicular control. In other words, a parallel parking system or a vehicle following system can easily be implemented on the designed prototype given appropriate modifications on the main control unit. This chapter focuses on the design and implementation of a fuzzy logic and a neural network based controller for realization of the wall-following task. Experimental results of their hardware implementations are presented. A comparative study of the two controller designs is performed, with emphasis on hardware resource requirement, maximum operational speed and trajectory tracking error. Discussions regarding the advantages and drawbacks of each controller are provided.

5.1 Fuzzy Logic Controller (FLC)

Fuzzy logic provides a means to design robust control systems that are free of analytical models. Base upon human expert knowledge, the fuzzy reasoning process involves fuzzifying crisp input signals, evaluating individual fuzzy rules in the rule base, aggregating rule outputs and computing a crisp control signal through mathematical calculations. To design an effective fuzzy logic controller, proper definitions on linguistic variables, fuzzy sets and fuzzy rules are required. These parameters are commonly chosen by domain experts based on intuition and experience.

5.1.1 FLC Design

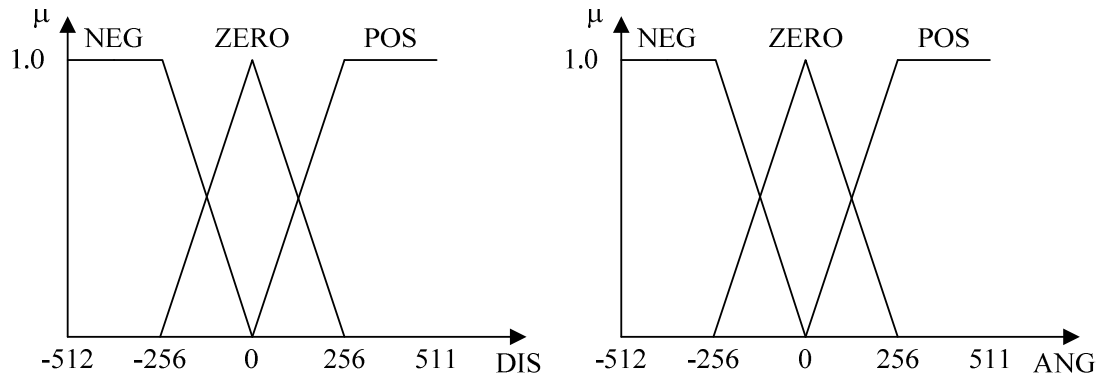
Linguistic Variables

The first step in constructing a fuzzy logic controller is to specify the control problem, with definitions of input and output linguistic variables. For the wall-following task, the objective is to maintain the vehicle at a predefined distance from the wall. One way to achieve this goal is by continuously adjusting the steering angles based on sensory data. In other words, by proper control of the steering angles, the controller should ensure that the vehicle's orientation is parallel to the wall and its back side sensor distance is at the predefined level. For realization of such a controller, two input variables, DIS and ANG, are used. The DIS variable is defined as DSBS – DW, where DW is the predefined desired distance from the wall. This variable specifies the relative distance between the back of the vehicle and the wall. The ANG variable is defined as DSBS – (DSFS + DF/2). DSBS – DSFS denotes the orientation of the vehicle with respect to the wall. If the forward direction of the vehicle is clear of obstacle, DSBS – (DSFS + DF/2) is approximately equal to DSBS – DSFS. On the other hand, if a wall is detected in front of the vehicle, DSBS – (DSFS + DF/2) will be smaller than

DSBS – DSFS, modifying the steering angle for an attempt to avoid the obstacle. Sensory data including DSBS, DSFS and DF are encoded into 10-bit digital signals by the ADC. However, to conserve computational resources, only the nine most significant bits are utilized for computing of the steering outputs. This effectively limits the three sensory data to a range of 0 to 511. Correspondingly, the DIS and ANG variables are confined to ranges of -511 to 511 and -766 to 511, respectively. For the same resource conservation reason, the universe of discourse of DIS and ANG are limited to ten bits, corresponding to the range of -512 to 511. The variable to be controlled is the steering output, ϕ. For simplicity, it is assigned the same universe of discourse as the input variables.

Fuzzy Sets

The next step in the fuzzy logic controller design cycle involves the selection of fuzzy sets for the description of fuzzy variables. The number of fuzzy sets in each universe of discourse is application specific. For problems with higher complexity, more fuzzy sets are required. The design of fuzzy sets also involves the choice of membership function type. They range from the linear triangular type to the nonlinear Gaussian type. To design a fuzzy logic controller for the wall-following task, the definition of three fuzzy sets for each input variable is sufficient. They are assigned linguistic terms NEG, ZERO and POS, denoting Negative, Zero and Positive. Membership functions of the triangular and the trapezoidal type are selected due to their linearity and simplicity. Figure 14 illustrates the input fuzzy sets employed.



(a) Linguistic variable DIS (b) Linguistic variable ANG

Figure 14: Fuzzy Sets for FLC Input Variables

Partition on the output universe of discourse is dependent on the inference model. For most control applications, the Sugeno Model is selected over the Mamdani Model as it is more compact and more computationally efficient. In these cases, the output membership functions are fuzzy singletons that take on membership value of 1 only at a single position along the universe of discourse. At all other locations, they assume a value of 0. For the current controller design, the Sugeno Model is selected

based on complexity and resource usage considerations. Nine output membership functions of the type singleton are defined, as shown in Figure 15. Their corresponding locations in the universe of discourse are summarized in Table 4. These values are approximated by human intuition and tuned by the trial and error approach.

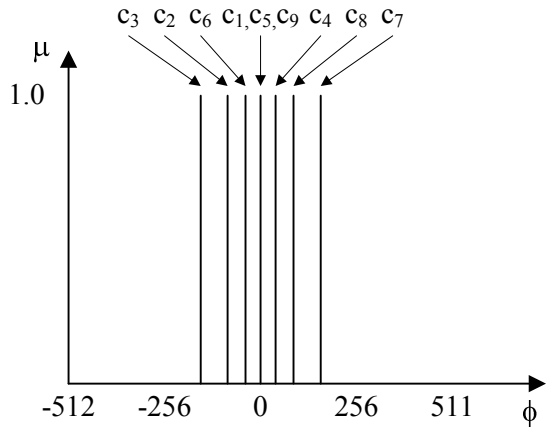


Figure 15: Fuzzy Sets for FLC Output Variable

Table 4: Consequents of FLC

Consequent	Value
c_1	0
c_2	-88
c_3	-160
c_4	40
c_5	0
c_6	-40
c_7	160
c_8	88
c_9	0

Fuzzy Rules

Human expert knowledge in a particular domain is incorporated into fuzzy if-then rules stored in the rule base. Based on intuition, nine fuzzy reasoning rules are developed for vehicle wall-following. They are summarized in Table 5.

Table 5: FLC Control Rule Base

		ANG		
		NEG	ZERO	POS
DIS	NEG	c_1	c_2	c_3
	ZERO	c_4	c_5	c_6
	POS	c_7	c_8	c_9

In detail, the rules can be expressed as follows:

R_1: IF DIS is NEG AND ANG is NEG THEN ϕ is c_1

R_2: IF DIS is NEG AND ANG is ZERO THEN ϕ is c_2

R_3: IF DIS is NEG AND ANG is POS THEN ϕ is c_3

R_4: IF DIS is ZERO AND ANG is NEG THEN ϕ is c_4

R_5: IF DIS is ZERO AND ANG is ZERO THEN ϕ is c_5

R_6: IF DIS is ZERO AND ANG is POS THEN ϕ is c_6

R_7: IF DIS is POS AND ANG is NEG THEN ϕ is c_7

R_8: IF DIS is POS AND ANG is ZERO THEN ϕ is c_8

R_9: IF DIS is POS AND ANG is POS THEN ϕ is c_9

where DIS, ANG and ϕ represent the distance input, the orientation input and the steering output, respectively. For evaluation of the multiple parts in the antecedents of these fuzzy rules, the minimum operator is utilized for the AND operation.

5.1.2 FLC Implementation

Sugeno Model based fuzzy inference systems are made up of three components: several fuzzifiers for computation of the degree of membership of crisp inputs in each membership function, an inference engine for generation of a fuzzy control output, and a mechanism to convert the fuzzy control output to a crisp control signal. To implement each of the three components in an intelligent fuzzy controller, various hardware modules are designed in VHDL based on two IEEE standard packages: numeric_std and std_logic_1164. These hardware units are located in the hardware FLC block, which is composed of fuzzifiers, min modules, multipliers, adders, a divider and a steering adjustment module. Each of them is described in detail in the following sections.

Fuzzifiers

The fuzzifier blocks fuzzify the input values originating from the central controller. In these blocks, the degree of membership of each input in each fuzzy set is evaluated. In other words, the intersections between the inputs and the membership functions are computed. Hardware based fuzzifiers are commonly designed utilizing a set of equations or a lookup table. The first approach is widely adopted for fuzzy sets that are characterized by linearity. For example, triangular and trapezoidal membership functions can be described using at most three linear equations. The latter method is usually employed for fuzzy sets with some degree of nonlinearity, such as Gaussian membership functions. Since the fuzzy sets used are of the linear type, the hardware fuzzifying unit is designed based on the first approach. The equations used are of the form y = ax + b where y represents the membership grade, a and b are coefficients specific to a particular equation, and x is the input. To further simplify the fuzzification process and conserve computational resources, the membership grades are scaled by 1023 such that any slope in the membership functions can be approximated by powers of two, as illustrated in Figure 16.

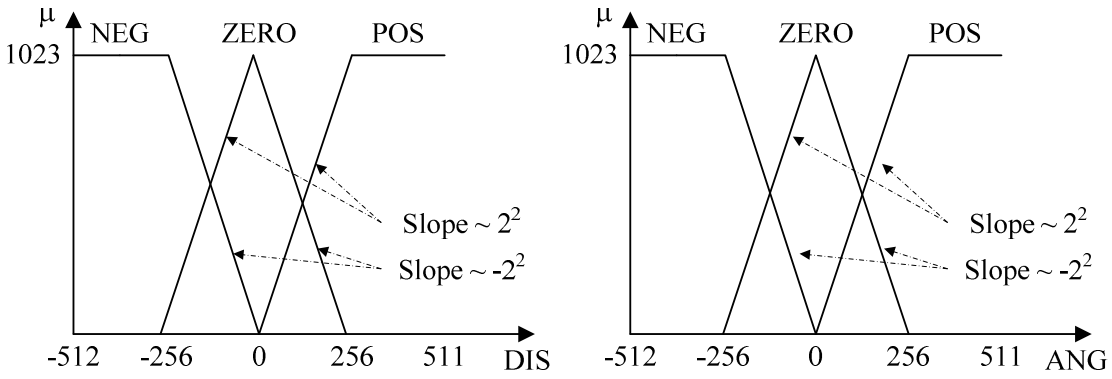


Figure 16: Scaled Fuzzy Sets for FLC Input Variables

Instead of utilizing computationally expensive multiplication and division circuitries, these mathematical operations are replaced by bit shifting logic where a multiplication or division of 4 is represented by shifting all the bit values by two spaces to the left or right respectively. It is important to note that this method is only an approximation, as the actual slopes in the diagonal segments are $\frac{1023}{256} = 3.996$ and $\frac{-1023}{256} = -3.996$, which is substituted by a value of 4 and -4. Figure 17 presents a flowchart with steps involved in finding the membership grade of an input value in the NEG fuzzy set. Bit shifting and complement calculation is required for computation of the membership grade. As an example, the binary representation of -250 is 11 0000 0110. With a shift of two bits to the left, the value becomes 00 0001 1000, which has a complement value of 11 1110 0111 corresponding to the membership grade of 999 in decimal. Note that the utilization of bit

shifting operators introduces a negligible amount of error. Compared to the approximation, the actual result is $\frac{1023}{256}(250) = 999.02$, with an error of 0.002%.

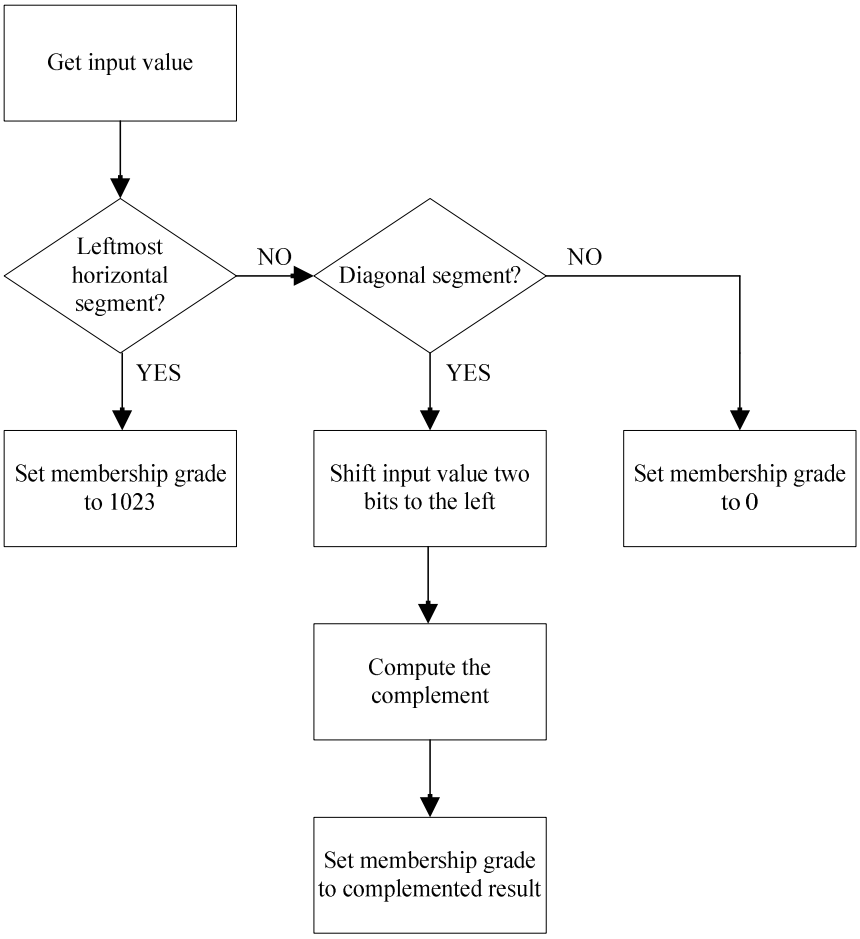


Figure 17: Flowchart for Calculation of the Degree of Membership in the NEG Fuzzy Set

With two input variables (DIS and ANG) and correspondingly three fuzzy sets to describe each of the variables, two fuzzifiers are utilized to generate a total of six membership grades (n1, n2, z1, z2, p1 and p2) representing the outputs of the fuzzifier modules, as shown in Figure 18. These outputs are connected to the inputs of the min modules.

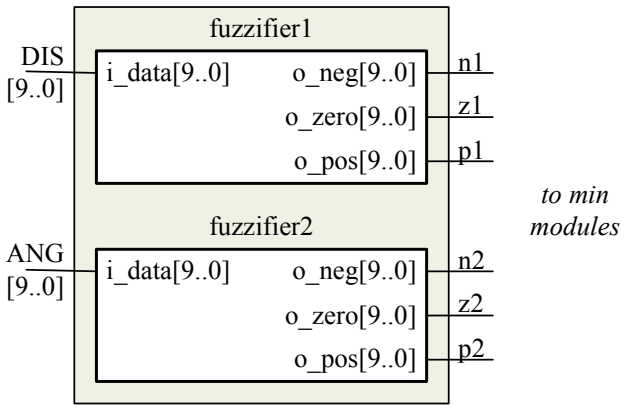


Figure 18: Fuzzifiers in Hardware FLC Block

Min Modules

Following the fuzzification step, the next stage is to realize the inference engine. In this phase, the antecedent of each rule is evaluated and the corresponding weighting parameter is computed. For a rule antecedent with multiple parts, a T-norm operator (in this specific case, the minimum operator) is utilized to attain the AND operation. The weighting parameter is then determined as the minimum membership grade of the input variables in each of the corresponding antecedent fuzzy set. For example, let the two inputs take on values of -250 and -256. Fuzzifying the two inputs then result in membership grades of 999, 24, 0 and 1023, 0, 0 corresponding to the degree of membership of the first and the second inputs in the NEG, ZERO, POS fuzzy sets. To determine the weighting parameter for R_1, the membership grades for the two inputs in the NEG fuzzy sets are compared. The degree of membership of the first input in the NEG fuzzy set is 999 while that of the second input is 1023. Thus, the minimum degree of membership is 999, representing the weighting parameter for R_1.

R_1: IF DIS is NEG AND ANG is NEG THEN ϕ is c_1

In hardware, the minimum operations are realized by comparators residing in the min modules. Each min module accepts two membership grades from the fuzzifier modules and compares them to determine the weighting parameter of each rule, as shown in Figure 19. With nine inference rules, nine weighting parameters m1 to m9 are determined. This information is then connected to the multipliers and utilized in further calculations for the overall crisp control output.

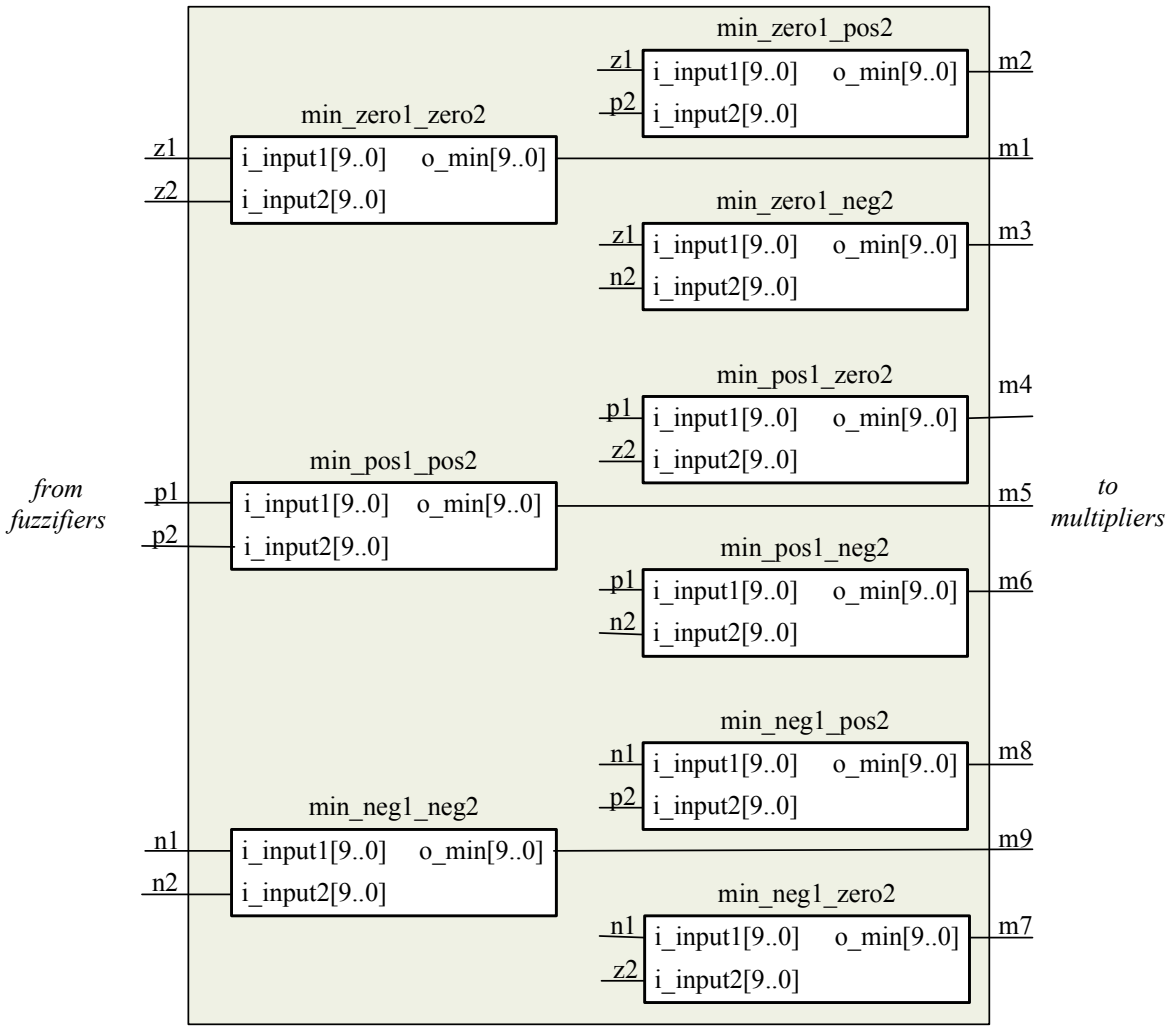


Figure 19: Min Modules in Hardware FLC Block

Adders, Multipliers and Dividers

The next part of the fuzzy logic controller computes the crisp control output utilizing weighting parameters found in the previous step. This involves calculation of the weighted average of individual rule output. With nine fuzzy rules, the weighted average calculation is governed by the following equation:

$$\hat{c} = \frac{\omega_1 c_1 + \omega_2 c_2 + \omega_3 c_3 + \omega_4 c_4 + \omega_5 c_5 + \omega_6 c_6 + \omega_7 c_7 + \omega_8 c_8 + \omega_9 c_9}{\omega_1 + \omega_2 + \omega_3 + \omega_4 + \omega_5 + \omega_6 + \omega_7 + \omega_8 + \omega_9} \qquad (13)$$

where c_i represents the consequent of rule i, ω_i is the weighting parameter for rule i and $\hat{c}$ is the overall crisp control output. In hardware, this can be realized by two-input multipliers, three-input adders and two-input dividers. Nine multipliers are employed for performing multiplications between the consequent and the weighting parameter of each rule. Four adders are then utilized to sum the

multiplication results, generating the numerator value for the division operation. To generate the denominator, four additional adders are employed for summation of the weighting parameters. Finally, a divider is utilized to calculate the control output signal by dividing the numerator by the denominator, as shown in Figure 20.

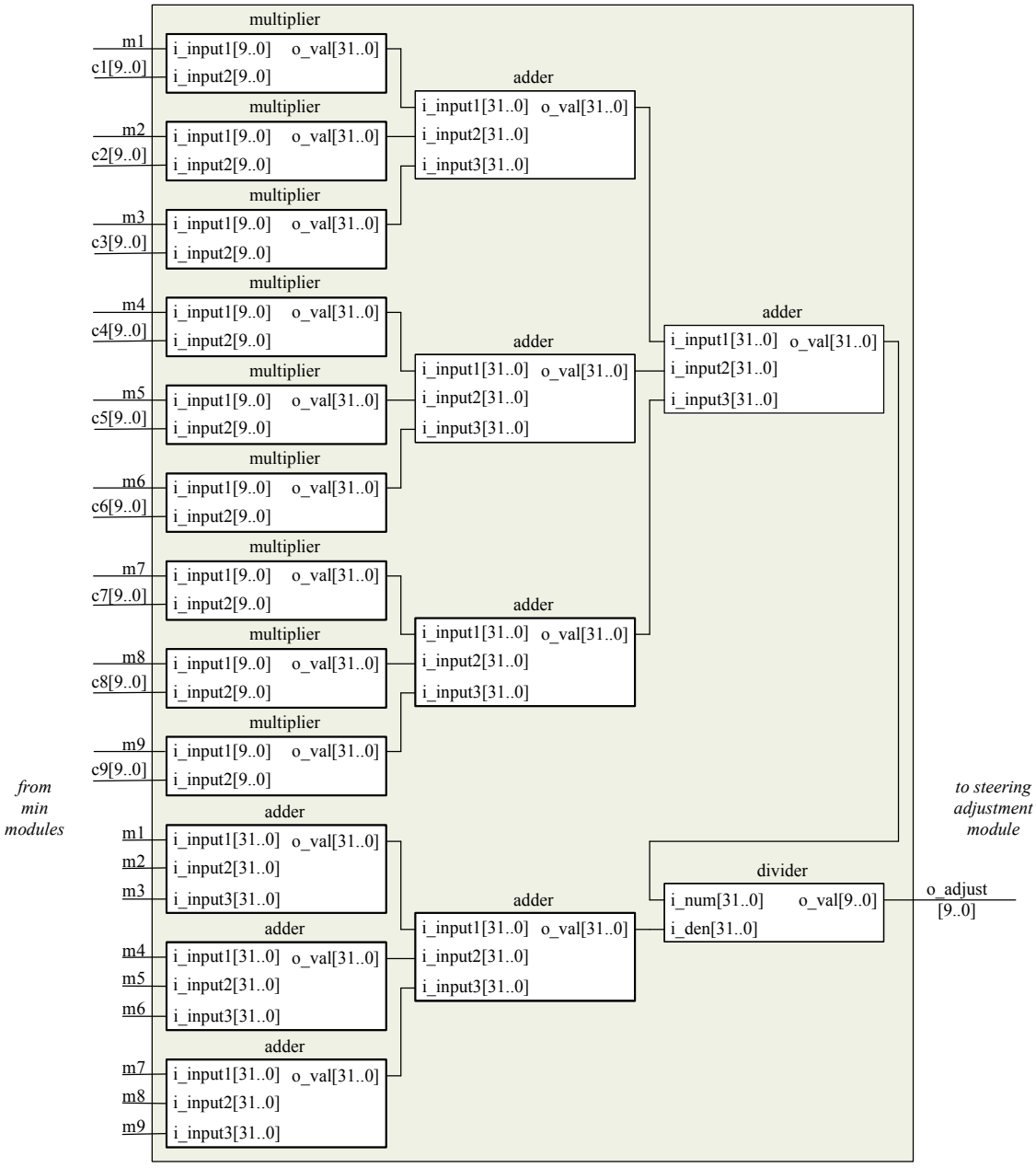


Figure 20: Multipliers, Adders and Divider in Hardware FLC Block

Steering Adjustment Block

The control output values are limited to a range of -512 to 511, corresponding to its universe of discourse. However, only fifteen discrete steering outputs with assigned values from 0 to 14 are defined in the steering mechanism. Thus, a steering adjustment block is employed to quantize the wide range of control outputs into fifteen values according to Table 6. Once the steering output value is selected, the information is sent to the steering mechanism, as shown in Figure 21.

Table 6: Steering Adjustment Mapping

Output Ranges	Steering Output Values
> 40	14
37 to 40	13
33 to 36	12
27 to 32	11
19 to 26	10
11 to 18	9
6 to10	8
-4 to 5	7
-9 To -5	6
-17 To -10	5
-25 to -18	4
-31 to -26	3
-35 to -32	2
-39 to -36	1
<= -40	0

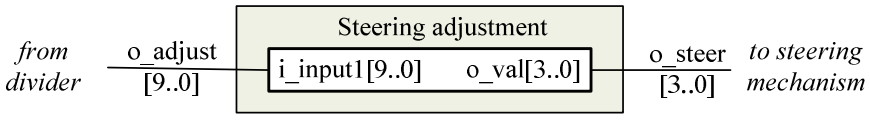


Figure 21: Steering Adjustment Module in Hardware FLC Block

5.2 Neural Network Controller (NNC)

Neural network, a structure composed of individual processing units joined by a large number of parallel interconnects, provides a means to develop a generalized control unit with fault tolerant

characteristics. Similar to fuzzy logic control, neural network control is also a model-free control technique which has the advantage of basing its decisions on past experiences. Neural network control involves calculating the weighted sum of individual processing inputs and computing the resultant processing unit output via a mapping function. The design of an effective neural network controller involves collection of representative training data, definition of an appropriate network structure and execution of the training process.

5.2.1 NNC Design

Training Patterns

Collection of representative training data is an important stage in the development of a neural network. Often, neural networks are only as good as their training sets. Results can be improved by utilizing a greater number of training patterns, which are sufficiently spaced within their operating ranges. One key issue in the data collection process is the selection of parameters to focus on. This is usually application-specific and depends on the control objective. For example, the objective is to determine the value of deceleration as a vehicle approaches a red traffic light. In this case, the inputs may be the distance to the light and the current speed of the vehicle while the output is the deceleration value. These input and output variables must be taken into consideration during the training process. As mentioned previously, the overall objective of the wall-following task is to maintain the vehicle at a predefined distance from the wall. This can be achieved by applying steering control based on sensory information. In this scenario, a training pattern should incorporate both sensory information and the ideal steering output. Hence, side sensor measurements, DSFS and DSBS, as well as the steering output, ϕ, are included in each training pattern. To closely mimic human driving behavior, training patterns with these parameters are gathered by recording the driving experiences of human operators, which can be collected through manual control of the prototype vehicular system via the control pad. It is possible to collect this training data using alternative methods.

The training data needs to be representative to ensure satisfactory neural network controller performance because these control units tend to be excellent interpolators but poor extrapolators. In other words, the controller excels for those cases in which the network has been trained while it performs poorly for those which are underrepresented by the training set. For the wall-following application, three different trajectories are designed for neural network training: straight, angled, and corner. Section 5.3.3 discusses these trajectories in detail. The general input operating ranges for the training set are 0 to 380 for DSFS and 80 to 337 for DSBS. Good performance is expected when the vehicle traverses the trained trajectories, corresponding to input values within the training ranges. On the contrary, poor performance may result from untrained scenarios.

Network Structure

Commonly used feed-forward type neural networks have one input layer, one output layer, and one or more hidden layers. The number of neurons in the input layer and the output layer correspond to the number of input and output parameters in each training pattern. However, the number of hidden layers and hidden neurons in each layer varies depending on the complexity of a given problem. A network with a small number of hidden neurons can be problematic as it may be insufficient for approximating the nonlinearities involved in that particular problem. On the other hand, a network with a large number of hidden neurons may over-parameterize hidden relationships in training patterns thus impairing its generalization capability. This effect is often termed as overtraining. Although experimental neural networks generally have several hidden layers, practical designs have at most one hidden layer due to the exponential increase in computational complexity added by each hidden layer [67]. For the application in discussion, the input layer is composed of two neurons corresponding to the sensory inputs while the output layer consists of one neuron, which is associated with the steering output. To minimize computational resources, only one hidden layer is employed. In terms of the number of neurons within the hidden layer, a trial and error approach is taken to obtain a suitable value, given that there are no set rules.

Another crucial parameter in the design of a neural network is the mapping function, or the activation function. Poor choice of activation function limits the performance of the network. The ideal mapping function is not only application-specific, but also takes into account the level of nonlinearity and complexity in the problem. As discussed previously, the steering mechanism recognizes fifteen different values in an assigned range of 0 to 14. To ensure that the direct control output from a neural network is bounded within this range, without an external scaling function, the activation function in the output layer must be chosen accordingly. Amongst the most commonly utilized mapping functions, only the linear function has output values outside of the range of -1 and 1. Thus, it is selected as the output activation function for the wall-following application. For the hidden layer, three different mapping function types are experimented on: linear, saturating linear, and hyperbolic tangent.

Network Training

Network training is essentially a process to optimize the associated weight parameter for each connection between two processing units such that the trained network assumes the required functionality and differences between the network and the desired outputs are minimized. Each weight in the network describes the strength of a specific connection. For instance, a weight of zero means that the two units are unconnected. There exist many learning algorithms for adjusting these weight parameters to properly train a neural network. For training sets consisting of both input and output pairs, the most widely used technique is supervised learning in which the network is trained by example. An input pattern is first presented to the input layer of the network and is propagated through subsequent layers until an output pattern is obtained in the output layer. An error value, known as network error, is then calculated by comparing the network output with that of the training pattern. The weights are adjusted as to minimize this error. This process continues until satisfactory

error value is obtained and the network has been trained with all the desired characteristics and functionalities. The gradient descent methodology modifies the weights in the direction of which the network error decreases most rapidly. BPL, the most commonly used algorithm for supervised learning, is based on this tactic. BPL sometimes suffers from its long training time in practical applications. A more attractive training algorithm is the Levenberg-Marquardt Learning technique, which combines Newton's Method with the gradient descent methodology utilized in BPL. It is superior to the BPL algorithm in terms of training times.

The neural network based wall-following controller is trained using supervised learning schemes with a set of input-output pair training patterns. The Levenberg-Marquardt Learning algorithm is selected for this purpose due to its rapid training time. To conserve hardware resources and development time, training of the network is performed off-line using Matlab and then, the trained network is implemented in the vehicle model for real-time experiments. Network training is performed on a few different network structures which differ in the number of hidden neurons and the activation function type. The best architecture, which produces the smallest network error, is chosen as the final design for hardware implementation. In the first set of experiments, the number of hidden neurons is varied. A hyperbolic tangent activation function is employed in the hidden layer and a linear one in the output layer. Once the optimal number of hidden neurons is defined, the second set of experiments is conducted to obtain the best activation function type for the hidden layer.

Table 7 summarizes the training results when using five, ten and fifteen hidden neurons. The mean square network errors after one thousand training cycles are presented. It is concluded that the optimal number of hidden neurons is ten. With an increase in the number of neurons from ten to fifteen, an increase in network error is observed due to the overtraining phenomena. Too many neurons result in memorization of training patterns, impairing the network's ability to generalize and interpolate for extraction of hidden relationships among input and output patterns. On the other hand, network error also increases with a decrease in number of neurons from ten to five. This is expected because with a small number of hidden neurons, the network structure is insufficient to approximate the level of nonlinearity in the given problem. Table 8 summarizes the training results when employing linear, saturating linear and hyperbolic tangent activation functions in the hidden layer. The network errors after one thousand training cycles are presented. Ten hidden neurons are utilized in all three cases. The linear function, which generates the largest network error, cannot adequately approximate the large amount of nonlinearities presented in the training set. The saturating linear function, which produces less error, provides a better approximation. Of the three selections, the hyperbolic tangent function generates the smallest error and hence, is ultimately employed in the hidden layer.

Table 7: Network Error using Five, Ten and Fifteen Hidden Neurons

Hidden Neurons	Mean Squared Network Error
5	1.5607
10	1.4421
15	1.4674

Table 8: Network Error for the use of Three Different Activation Functions

Activation Function	Mean Squared Network Error
Linear	1.8966
Saturating Linear	1. 4766
Hyperbolic Tangent	1. 4421

Figure 22 presents the final network design after training. Neurons are organized in a feed-forward manner with two, ten and one neurons in the input, hidden and output layers, respectively. The activation function of the hidden layer is hyperbolic tangent while that of the output layer is linear.

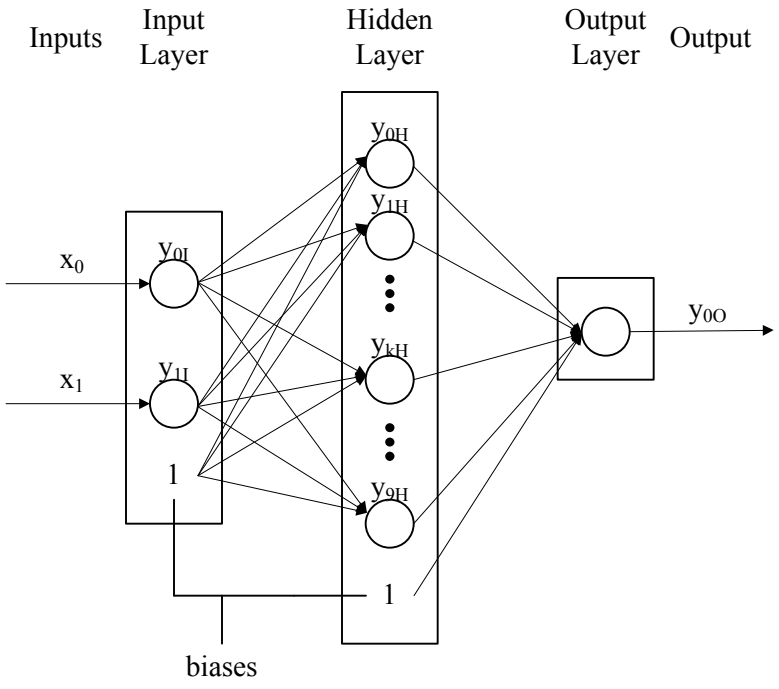


Figure 22: Final Neural Network Design

The input-output relationship of the network can be described by the following equations:

Input Layer

$$y_{0I} = x_0 \tag{14}$$

$$y_{1I} = x_1 \tag{15}$$

Hidden Layer

$$y_{kH} = f(\omega_{kH0I}\, y_{0I} + \omega_{kH1I}\, y_{1I} + \theta_{kH}) \quad \text{for k = 0, 1, 2, …, 9} \tag{16}$$

$$f(x) = \frac{2}{1+e^{-2x}} - 1 \tag{17}$$

Output Layer

$$\begin{aligned} y_{0O} = {} & \omega_{0O0H}\, y_{0H} + \omega_{0O1H}\, y_{1H} + \omega_{0O2H}\, y_{2H} + \omega_{0O3H}\, y_{3H} + \omega_{0O4H}\, y_{4H} \\ & + \omega_{0O5H}\, y_{5H} + \omega_{0O6H}\, y_{6H} + \omega_{0O7H}\, y_{7H} + \omega_{0O8H}\, y_{8H} + \omega_{0O9H}\, y_{9H} + \theta_{0O} \end{aligned} \tag{18}$$

The two inputs of the network, x_0 and x_1, represent the two sensory inputs, DSFS and DSBS, respectively. The network output, y_{0O} represents the steering output, ϕ. The function $f(.)$ is a hyperbolic tangent activation function, given by (17). ωs and θs represent the adjustable weights and biases obtained from training, with values summarized in Table 9. All weights are denoted as $\omega_{\alpha\beta\gamma\delta}$, representing the strength of the connection from neuron γ in the δ layer to neuron α in the β layer. For example, ω_{9H0I} represents the strength of the link from neuron 0 in the input layer to neuron 9 in the hidden layer. Bias values denoted as $\theta_{\alpha\beta}$ follows the same notation. As an example, θ_{0O} represents the bias value associated with neuron 0 in the output layer.

Table 9: Trained Weights of NNC

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
ω_{0H0I}	-0.1061	ω_{0H1I}	0.0553	θ_{0H}	16.3239	ω_{0O0H}	-0.4984
ω_{1H0I}	0.8513	ω_{1H1I}	-0.6289	θ_{1H}	-7.6035	ω_{0O1H}	42.8902
ω_{2H0I}	-0.3400	ω_{2H1I}	0.3988	θ_{2H}	18.3682	ω_{0O2H}	-0.3560
ω_{3H0I}	-0.0772	ω_{3H1I}	0.2347	θ_{3H}	-15.9546	ω_{0O3H}	-0.2522
ω_{4H0I}	0.0789	ω_{4H1I}	-0.0146	θ_{4H}	-7.1801	ω_{0O4H}	1.1693
ω_{5H0I}	-0.0292	ω_{5H1I}	-0.0060	θ_{5H}	7.1676	ω_{0O5H}	-1.2703
ω_{6H0I}	0.1235	ω_{6H1I}	-0.3361	θ_{6H}	52.4580	ω_{0O6H}	-0.1684
ω_{7H0I}	0.0184	ω_{7H1I}	0.0360	θ_{7H}	-6.2948	ω_{0O7H}	1.1598
ω_{8H0I}	0.8559	ω_{8H1I}	-0.6323	θ_{8H}	-7.6365	ω_{0O8H}	-42.5374
ω_{9H0I}	-0.0355	ω_{9H1I}	0.0653	θ_{9H}	-16.6307	ω_{0O9H}	-1.6749
						θ_{0O}	3.4157

5.2.2 Manual Controller (MC) Implementation

For development of a neural network controller that closely emulates human driving behavior, the manual mode of operation is utilized for data collection purposes. In this mode, the steering outputs are controlled by a human operator through the use of a control pad. It is solely based on the judgment of the operator, independent of collected sensory data. Sensory inputs and steering outputs are recorded during manual driving experiences and then, employed in the neural network training phase. According to Section 4.4, the control pad consists of a number of switches with mechanical contacts. The switches are normally opened, which corresponds to grounded outputs while pressed buttons correspond to high output signal. However, clean transitions between low and high logic levels are difficult to attain with commonly available switches. During transitions between open and closed states, a series of spikes in the signal level is expected, which is referred to as a bouncing problem. Figure 23 presents the output waveform of a switch demonstrating the bouncing situation.

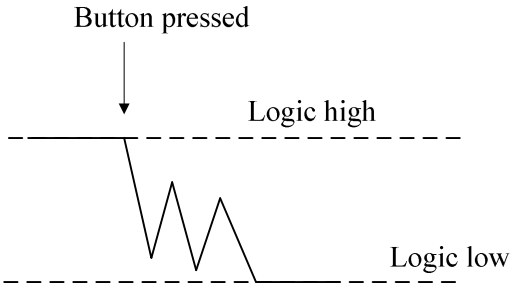


Figure 23: Sample Output Waveform of a Switch

In digital design, the bouncing issue can affect system performance. Instead of a single detection of the high or low state, multiple oscillating signals are detected. This may be falsely interpreted as repeated switching by the controller. To correct this problem, de-bouncing circuitries are required. One of the simplest methodologies for de-bouncing is the time-delay approach. By introducing additional delays to the design, the detection of the bounces is avoided. The length of the delay is generally application dependent as short delays fail to eliminate the bouncing problem while lengthy delays results in sluggish responses. In addition to a de-bouncing circuit based upon the time-delay approach, the manual controller implements steering control using adders and comparators. On reset, the steering output of the controller defaults to 7, representing a steering angle of 0^o according to Table 3. Upon activation of the right button, the controller decrements the steering output down to a minimum of zero, which represents a steering angle of 30^o to the right. Similarly, the steering output increments with each push of the left button up to a maximum of fourteen, which represents an angle of 30^o to the left.

5.2.3 NNC Implementation

Implementation of a neural network is essentially the realization of a set of mathematical equations for signal propagation from the input to the output layer. The propagation process has two steps. The hidden layer neuron outputs are first computed by passing the weighted sum of their corresponding inputs through a mapping function. The output layer neuron then computes its result as the weighted sum of its inputs through another mapping function.

Signal Representation

One common issue in neural network implementation is signal representation. Performance of neural networks is highly dependent on the data representation scheme. Conventional integer representation is inadequate for representing floating point weights generated by Matlab. Precision is significantly reduced assuming no additional scaling arithmetic units. To represent the trained weights, two signal representations, namely floating point representation and fixed point representation, are investigated. In floating point representation, the location of the radix point, separating the integer part and the

fractional part of a number, can vary allowing for wide range and high precision. Because this representation closely approximates real numbers, development time and complexity are reduced. However, it is impractical to use floating point representation in hardware implementation as they require large amounts of computational resources. Fixed point arithmetic provides a compromise between integer and floating point representations. The former significantly reduces precision while the latter significantly increases computational resources. In fixed point representation, the location of the radix point is fixed, assigning a fixed number of digits for the integer part and the fractional part of a real number, respectively. With limited range and precision, it is difficult to represent very large and very small numbers in the fixed point notation. In other words, it is prone to overflow and precision loss issues. However, the arithmetic logic is simpler compared to that of the floating point notation. In hardware implementation, this translates to less power consumption, less processing time, and lower cost. To take advantage of parallelism in neural networks without sacrificing valuable computational resources, fixed point representation is utilized in the neural network controller design. All trained weights are converted to fixed point representation in Matlab prior to hardware implementation. To avoid undesired precision loss and overflow issues, a 20-bit fixed point representation is chosen with the first eight bits representing the integer portion of a numeric value and the last twelve bits representing the fraction portion.

Hardware Modules

To implement an intelligent neural network controller, various hardware modules are designed in VHDL. Currently, standard VHDL packages have no support for fixed point arithmetic. To implement the NNC in fixed point notation, an open source fixed point package [68] is utilized. It is built on an existing IEEE standard package, numeric_std, employing built-in data types and attributes. Fixed point arithmetic operations in the open source package include comparison, addition, subtraction and multiplication. However, division operations are not synthesizable. The hardware NNC module is composed of a number of multipliers, adders, act_fun modules and sensor mapping blocks, which are described in the following sections.

Sensor Mapping Block

The neural network controller was trained using results from a manually controlled vehicle performing wall-following at a single predefined distance. However, it is often necessary to achieve wall-following at other distances. Instead of training the network with all possible wall-following distances, it is easier to utilize sensor mapping blocks. The sensor mapping blocks work by converting the desired wall-following distance to that of the trained data, and correspondingly change all the outputs of the neural network to match that of the desired distance. For example, the vehicle training data is performed for a distance of 13 cm. When the vehicle is in the autonomous mode and the desired wall-following distance is 13 cm, the sensor mapping blocks are not necessary. However, if 10 cm is selected, the sensor mapping blocks adjust the sensor data by 3 cm, allowing the neural network to apply the training data for the 13 cm trajectory into the selected 10 cm following distance, effectively shifting the path of the vehicle by 3 cm closer to the wall. The simplest method in the realization of a sensor mapping block is by means of a LUT. A single entry in the LUT consists of

two values, one for the actual sensor value and another for the mapped sensor value. For example, if the network is trained at a wall-following distance of 13 cm with a sensor value of 217 and the desired wall-following distance is 10 cm with a sensor value of 288, then 288 and 217 constitutes an entry in the LUT. When the sensor detects an input of 217, the LUT is utilized to interpret the sensor reading as 288. The LUT contains 512 entries, with one entry for mapping each of the 512 possible sensor values in the range of 0 to 511. To determine the effective sensor values for the two input variables, DSFS and DSBS, two sensor mapping blocks are utilized, as shown in Figure 24. They reside in the dedicated memory block of the FPGA.

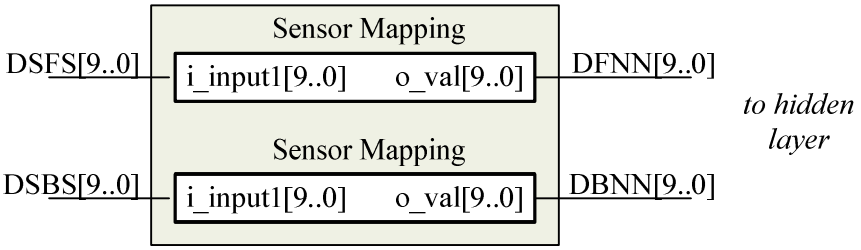


Figure 24: Sensor Mapping Modules in Hardware NNC Block

Hidden Layer

To generate the output signals of the hidden neurons, twenty multipliers, ten adders and ten act_fun modules are utilized, as shown in Figure 25. The multipliers with two input channels are employed to take the product of the mapped input sensory values and their associated weights. The adders, with three input signals, are then utilized to sum the multiplied signals and their corresponding bias values. The act_fun modules realize the hyperbolic activation function given in (17). Each module maps the weighted sum calculated by the multipliers and the adders to a single numeric value.

Efficient hardware implementation of the hyperbolic tangent activation function is a challenge due to the required exponential functions. Two approaches are commonly used in the realization of such nonlinear mapping functions. The first approach represents the activation function in a set of equations [69]. These equations can be of first-order or of a higher-order. The first-order approximation uses less hardware resources while high-order approximations produce higher precision. Linearization of the activation function results in a set of first-order equations of the form $y = ax + b$, where x represents the input to the activation function and y represents the output. To obtain the correct output value, comparators are required to select the appropriate equation. A multiplier and an adder are then utilized to implement the chosen equation. In general, multiplication circuitries are required to compute the ax term. Nevertheless, with appropriate chosen coefficients in powers of two, they can be eliminated and replaced by bit shifting operators, minimizing the amount of computational resources required. An example of a higher-order approximation is the quadratic equation $y = ax^2 + bx + c$. This representation can potentially produce more accurate results at the expense of hardware area since additional multipliers and adders are required to implement this

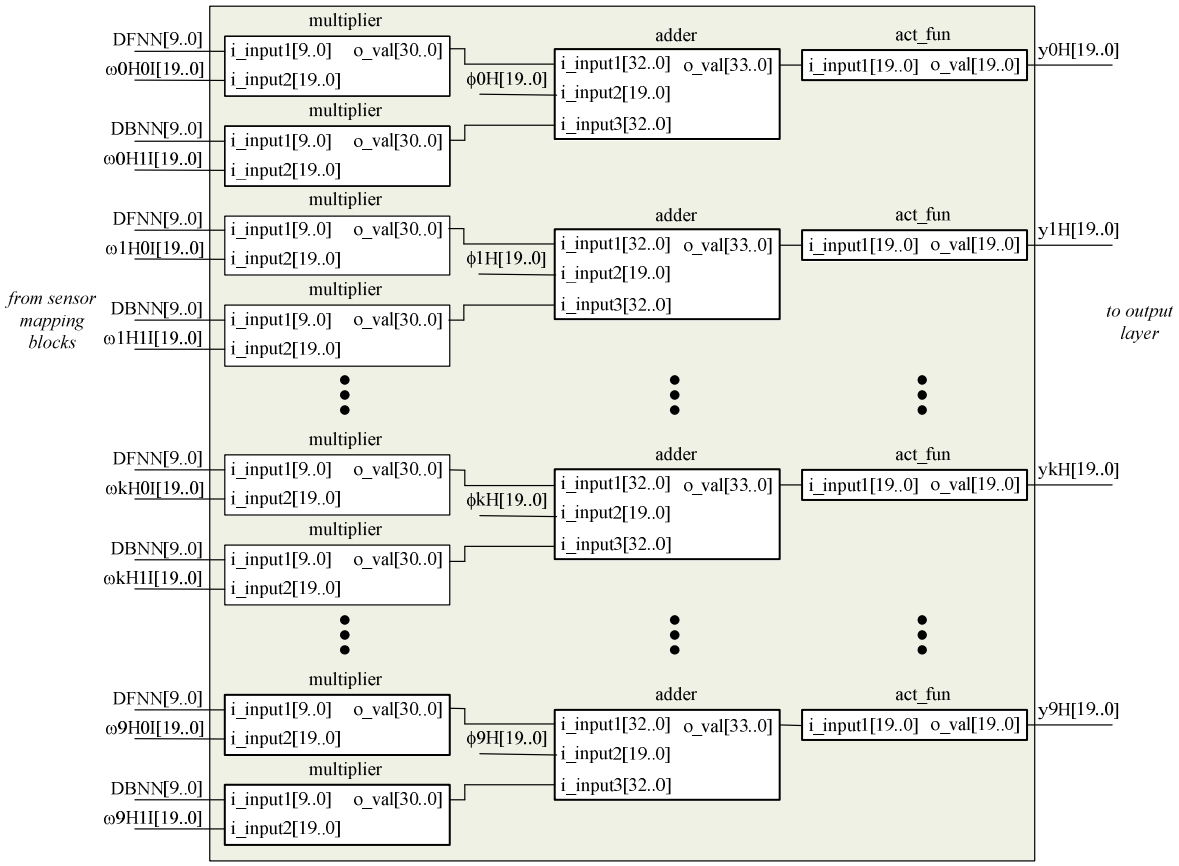


Figure 25: Hidden Layer in Hardware NNC Block

equation. The second way to approximate the activation function involves setting up a look-up table (LUT) [70]. Uniform samples of the activation function are stored in a table format. Rather than mathematical calculations, a read operation is performed to obtain the output value. The main advantage of this approximation technique is simplicity and flexibility. However, the main drawback lies in the large amount of LEs or memory units required. To realize the hyperbolic tangent activation function, a LUT with 141 entries is designed. The table consists of output values corresponding to inputs in the range of -3.5 to 3.5. Numerical inputs less than -3.5 and above 3.5 results in output values of -1 and 1, respectively. Figure 26 presents the hyperbolic activation function and its approximations based upon the LUT generated. It is seen that the approximated values closely follow the actual outputs.

Output Layer

To compute the network output according to equation (18), ten multipliers and an adder are used, as shown in Figure 27. Since the activation function is linear, additional hardware circuitry for its

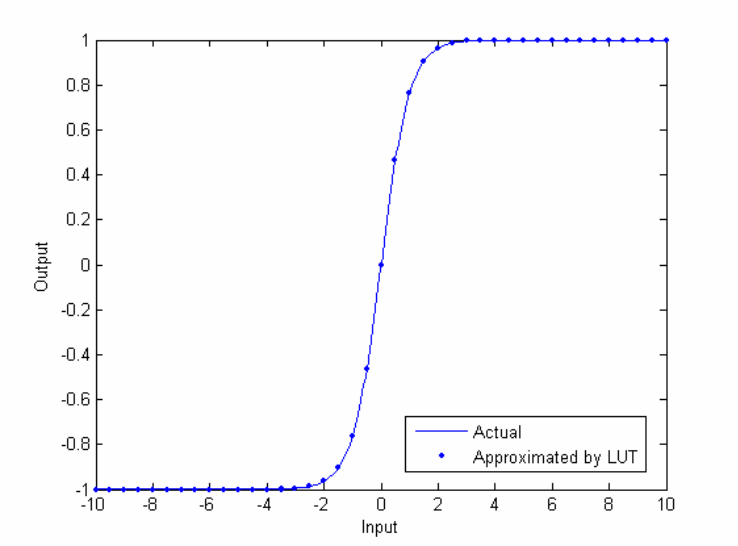


Figure 26: Actual and Approximated Hyperbolic Tangent Activation Function

implementation is unnecessary. The multipliers, which are identical to those employed in the hidden layer, have two input channels. One channel is assigned to the output of a given hidden neuron while the other is allocated for its corresponding weight. The product signals then serve as inputs to the adder whose output is the final output signal. The neural network is trained by patterns with

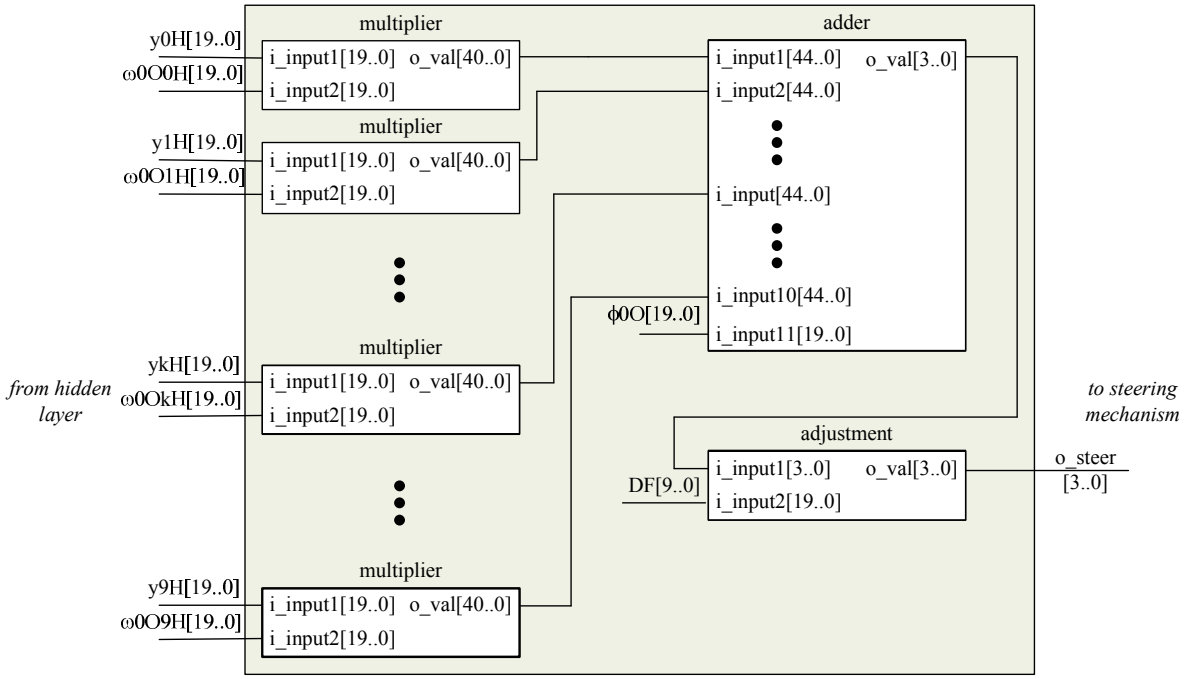


Figure 27: Output Layer in Hardware NNC Block

information from the side front and back sensors. However, to follow walls with abrupt changes, front sensor information is often required. Without this information, the vehicle will not have sufficient time to adjust its steering wheels upon recognition of the change. To resolve this issue without re-collection of training data, a simple adjustment rule is utilized. In cases where the front sensor detects an obstacle in close proximity, the steering output is adjusted such that the vehicle over-steers (utilize a larger steering angle compared to an obstacle-free situation) as an attempt to avoid the obstacle, thereby safely following the wall.

5.3 Performance Evaluation

Controller design effectiveness is generally measured by the metrics of accuracy, speed, and resource requirements. However, optimizing according to these criteria requires balancing tradeoffs. For example, performance is enhanced often at expense of additional resources while an improvement in speed may result in accuracy reduction.

5.3.1 Computational Resources and Operational Speed

Computational resources and operational speed are two vital factors in determining hardware cost. The ideal scenario utilizes the minimum amount of resources while maximizing speed.

Computational Resources

Computational resources in FPGAs are categorized as follows: Logical Element (LE), Random Access Memory (RAM), Digital Signal Processor (DSP) and Input Output Element (IOE). A LE is the smallest logic unit in FPGAs. By adjusting interconnects between LEs, both combinational and sequential logic can be realized. Theoretically, any design can be implemented utilizing only LEs. However, in practice, for efficient use of hardware area, blocks dedicated for specific purposes are also employed. These blocks tend to be more efficient because they are optimized for performing specific tasks. For example, RAM blocks are memory bits allocated for information storage while DSP blocks are designed for arithmetic operations. Frequently, the measure of computational resources and hardware utilization efficiency are based upon three of the four resources types previously mentioned. The number of LEs, quantity of memory bits, and amount of DSPs are metrics used to compare different designs. A design utilizing a smaller number of these elements is considered more computationally efficient than one that uses more. Table 10 summarizes the amount of computational resources required for the FLC and the NNC, respectively.

Table 10: Computational Resources Comparison

	FLC	NNC
LE (% of total)	17.34	38.22
Memory (% of total)	0	1.11
DSP (% of total)	37.50	41.67

Looking at Table 10, the FLC requires considerably less LE compared to the NNC. This difference is mainly due to the implementation methodology of the activation function in the NNC. Currently, the activation function LUT with 141 entries is realized using LEs. One way to reduce the amount of required resources is to decrease the number of entries in the table potentially at the expense of reduced precision. Another method is to relocate the LUT into dedicated memory blocks, as they provide higher efficiency for information storage. With these two means, the variance between the FLC and the NNC in terms of LEs can be reduced significantly. Aside from the number of LEs, the NNC also requires more memory units due to the additional sensor mapping blocks. LUTs in these blocks are utilized for mapping of sensor values to provide the vehicle with the ability to follow the wall at any predefined distance. In terms of DSP usage, the FLC requires 18 units consuming 37.50% of total available resources while the NNC requires 20 units consuming 41.67% of resources. This insignificant difference illustrates that both implementations use a comparable number of multiplication and addition operations.

Operational Speed

The operational speed of a design is constrained by its maximum clock frequency (Fmax), which is generally determined by the longest propagation time of a signal between two flip-flops. If the system clock is set above this frequency, propagation of any signal along this slowest path cannot reach its destination in time, causing undesirable behavior such as data loss, data corruption or even system failure. The value for Fmax can generally be obtained from the compilation reports of most design tools such as Quartus [71]. Utilizing this technique, the Fmax value for the FLC and the NNC are found to be 17.08 MHz and 45.47 MHz, respectively. In other words, the system clock of the NNC can operate approximately two times as fast as that of the FLC. The speed measurement is also constrained by the number of clock cycles required for a signal to propagate from an input channel to an output channel of the design, commonly termed as latency. Essentially, Fmax represents the shortest length of a single clock cycle and depending on the number of these cycles required for output generation, the overall speed of the design varies. One method to obtain the latency value for a particular design is through the use of a functional simulation. By providing the simulator with a set of input signals, the latency value is determined as the number of clock cycles required for generation of the corresponding output signal. Figure 28 and Figure 29 present two screen shots associated with the functional simulation results of the FLC and the NNC, respectively.

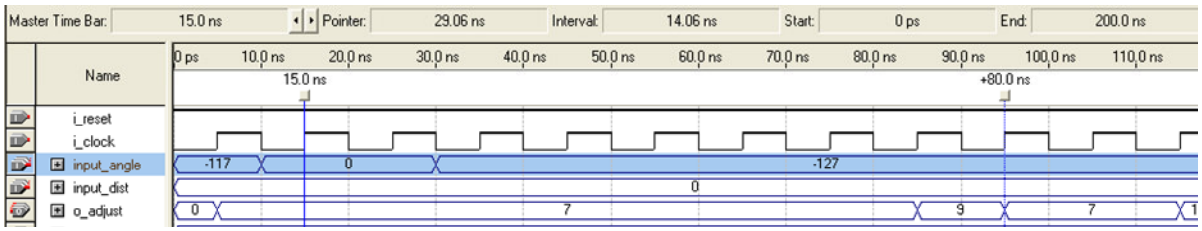


Figure 28: FLC Functional Simulation Results

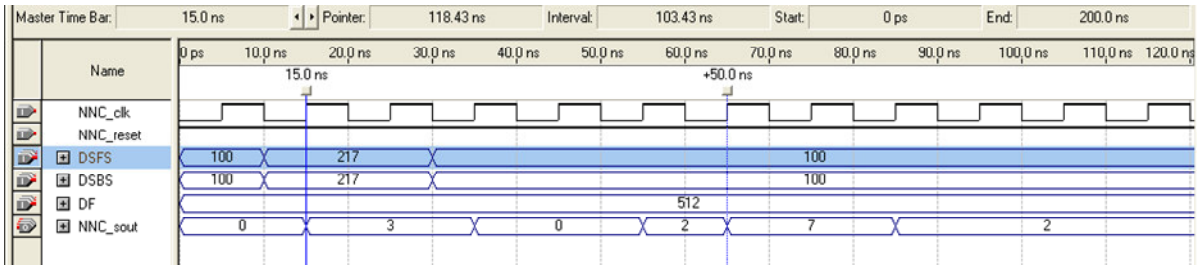


Figure 29: NNC Functional Simulation Results

For the FLC, the ideal scenario occurs when both input signals are zero, representing perfect alignment of the vehicle and the wall at the predefined distance. The desired steering output signal in this particular scenario is seven, corresponding to a steering angle of 0^{o} according to Table 3. As illustrated in Figure 28, the desired steering output of seven is generated 80 ns or eight clock cycles after setting the two inputs to zero. Thus, it can be concluded that the FLC has a latency of 8 clock cycles. For the NNC, the ideal scenario occurs when both input sensory values are "217" (in decimal representation), which maps to the predefined distance of 13 cm. The desired steering output signal in this situation is seven. As illustrated in Figure 29, the desired steering output is achieved 50 ns or five clock cycles after adjusting the two inputs to the predefined distance. Thus, the NNC has a latency of is 5 clock cycles, a reduction of approximately 37% from that of the FLC. With identical clock frequencies, a higher latency value means longer output generation time. Thus, given the same clock frequency, the NNC can generate outputs faster compared to the FLC.

In general, a high Fmax along with a small latency is desired. A design with a higher Fmax value but requiring a significant number of clock cycles to generate its output is only comparable in speed to a design with a lower Fmax requiring fewer cycles. To combine these two measures, the throughput time of a design is often calculated. The throughput time (TT) is governed by the following equation:

$$TT = \frac{Latency}{F\ \max} \tag{19}$$

It identifies the amount of time for a signal to propagate from an input to an output channel given the clock is set to its highest frequency, Fmax. In other words, it represents the shortest amount of time

an output can be generated without undesirable effects. In terms of throughput time, the FLC and the NNC require 468.38 ns and 109.96 ns, respectively. As expected, with smaller latency and higher Fmax, the NNC requires less TT, providing an output of approximately four times faster than that of the FLC when the clock frequency is set to its maximum value. This significant time difference is mainly due to two reasons. Firstly, the FLC utilizes a division operation, which is slow in nature resulting in a low Fmax value and hence, long TT. Secondly, the small number of hidden neurons facilitates a high Fmax while the small number of hidden layers and parallel nature of the NNC limit the number of input to output cycles required resulting in a short TT. Note that this deviation can be reduced significantly with more efficient implementation of the divisor in the FLC. Table 11 summarizes the maximum clock frequency, the latency and the throughput time for the FLC and the NNC. In general, the NNC provides higher bandwidth compared to the FLC.

Table 11: Operational Speed Comparison

	FLC	NNC
Fmax (MHz)	17.08	45.47
Latency (Clock Cycles)	8	5
TT (ns)	468.38	109.96

5.3.2 Experimental Results

Controller Validation and Comparison

Prior to integrating the two intelligent controllers with the prototype system, their corresponding hardware implementations are validated. Control surfaces (three dimensional surfaces illustrating the mapping from two sensory inputs to the steering output) generated from Matlab simulations are compared to those generated by a soft core microprocessor NIOS II. 676 different scenarios uniformly distributed in the input operating ranges of 0 to 511 are investigated. For generation of a control surface utilizing NIOS II, the microprocessor core and one of the two intelligent controllers are programmed onto the FPGA. C code is generated such that the microprocessor successively provides the intelligent controller with one of the 676 test patterns. To collect the corresponding output value, the input channel of the microprocessor is connected to the output channel of the intelligent controller, as shown in Figure 30.

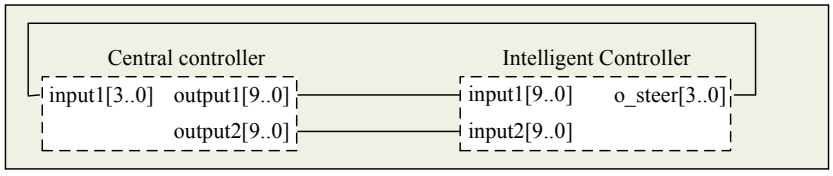


Figure 30: Validation Setup

Fuzzy Logic Controller Validation

Figure 31 and Figure 32 present the two control surfaces of the FLC, in which the NIOS II-based surface with hardware behavior closely approximates that from the Matlab simulation. This conclusion confirms the overall successfulness of the hardware FLC implementation.

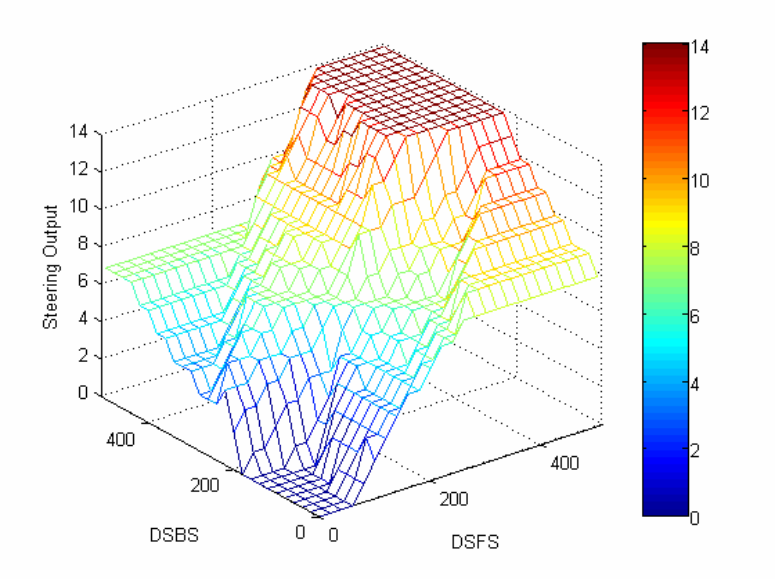


Figure 31: FLC Control Surface Generated by NIOS II

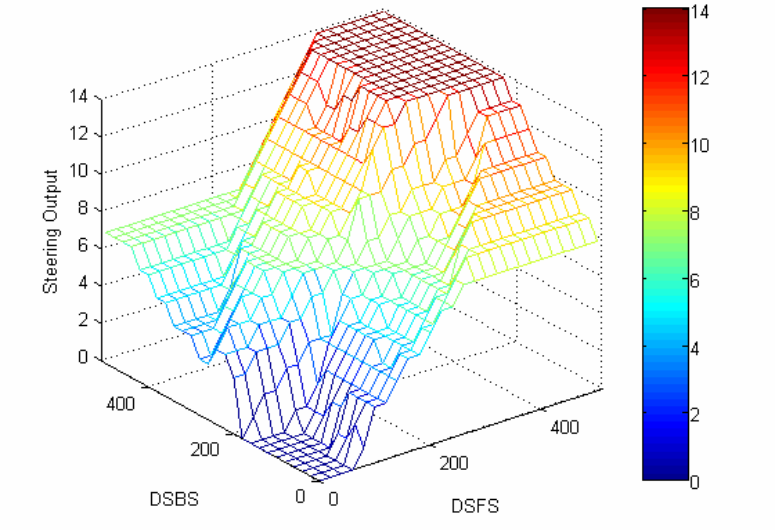


Figure 32: FLC Control Surface Generated by Matlab

An error surface, shown in Figure 33, is formed by subtracting the Matlab control surfaces from the NIOS II surface. The maximum error between the two control surfaces is 1. This is mainly due to the use of an integer divider in the hardware FLC block for realization of (13), which truncates the fractional portion of the floating point quotients, if any. For example, a crisp fuzzy inference output of 5.9 will generate a steering output of 8 in Matlab. However, in the hardware implementation, which uses an integer divider, the value of 5.9 will first be truncated into 5, resulting in a steering output value of 7. Note that if the Matlab generated output differs from that of the NIOS II, there is always a difference of 1, with the Matlab generated output being the larger value. To quantize the amount of output error, the mean squared error of the error surface is computed to be 0.1080.

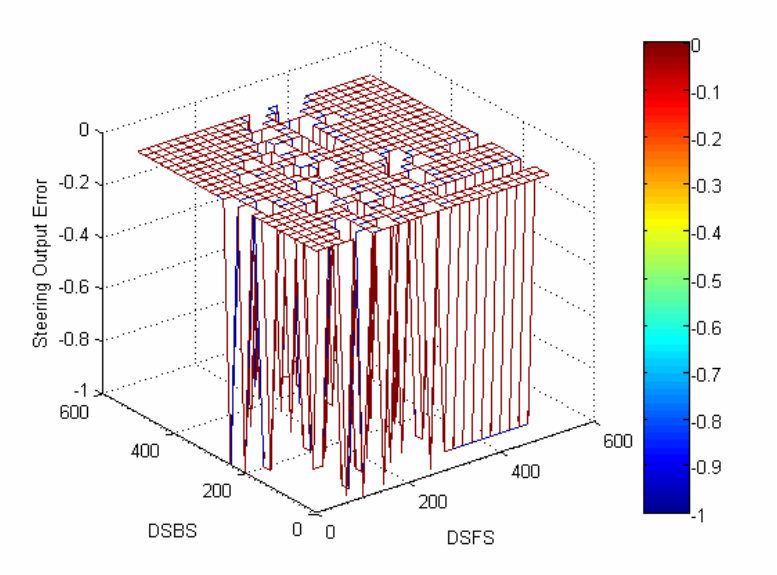


Figure 33: FLC Error Surface

Neural Network Controller Validation

To verify the hardware implementation of the NNC, Figure 34 and Figure 35 present the NIOS II-based control surface and the Matlab-based control surface, respectively. These control surfaces are generated using the same input values as in the FLC validation. The two NNC surfaces exhibit similarities validating the hardware implementation.

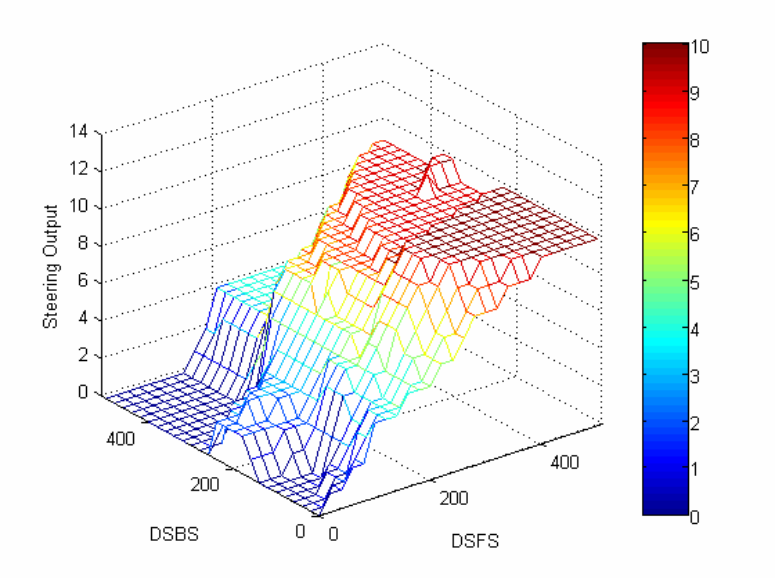


Figure 34: NNC Control Surface Generated by NIOS II

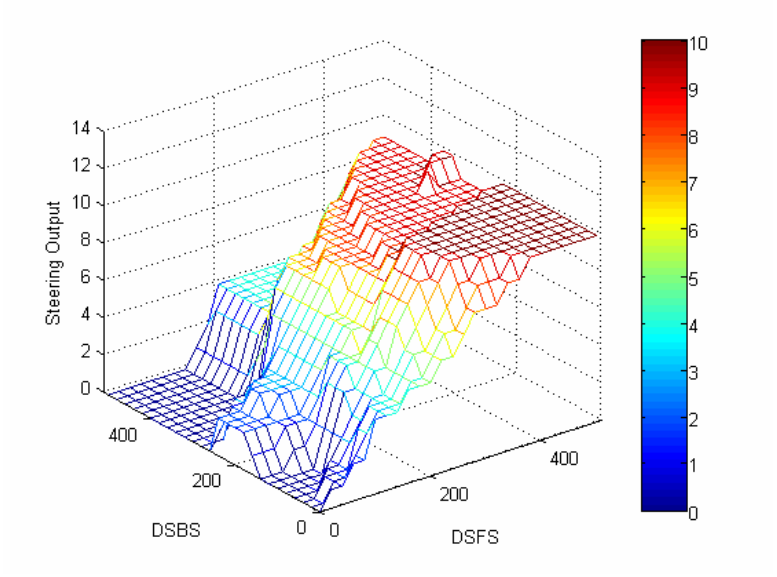


Figure 35: NNC Control Surface Generated by Matlab

Figure 36 illustrates the error surface of the NNC, representing the differences between the NIOS II and the Matlab results. Upon close examination of the surface and its corresponding contour plot shown in Figure 37, the maximum error in steering outputs is found to be 2. Depending on the number of occurrences and the location of the error on the error surface, this may or may not be significant. If the difference of 2 constantly changes an output from left steering to right steering, then the error becomes significant. However, if the difference results only in a change in magnitude

of the steering output in the same direction, it is insignificant. Based on Figure 37, the difference of 2 occurs only in one specific location. In this case, Matlab simulation provides a steering output of 7, corresponding to a steering angle of 0^o. In comparison, the hardware NNC block results in a steering output of 5, corresponding to a steering angle of 12^o to the right. This variation is mainly due to quantization error in the use of the 20-bit fixed point signal representation. To avoid such a difference, more digits in the fixed point representation are required, which directly translates into higher computational resource requirements. Since the discrepancy occurs only in one location and

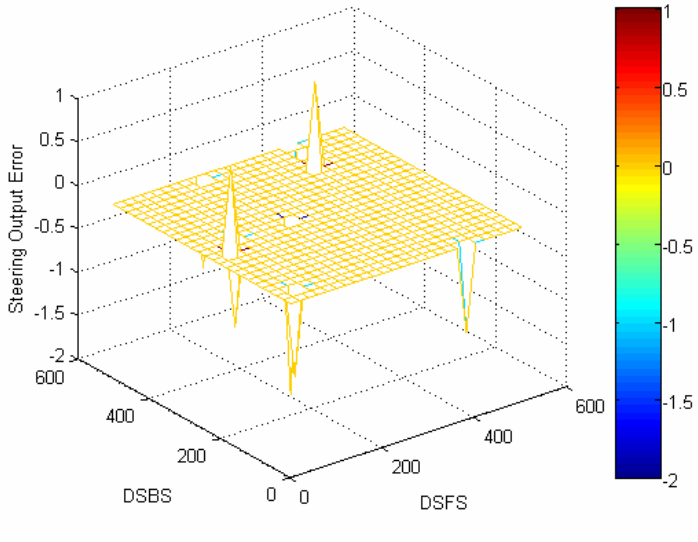


Figure 36: NNC Error Surface

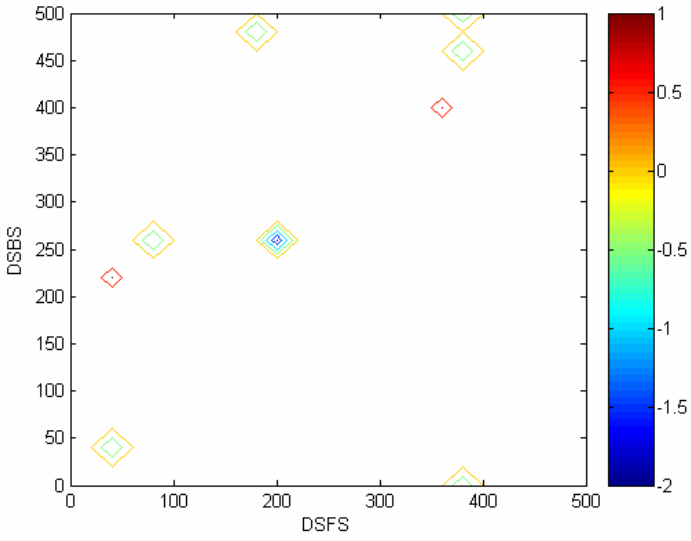


Figure 37: NNC Error Contour Plot

sensory data is noisy in nature, coupled with the fact that thousands of computations occur per second, this specific error is insignificant and the number of digits in the signal representation is not changed. Overall, the mean squared error of the error surface is calculated to be 0.0178, which is relatively small.

Controllers Comparison

The control surfaces for the FLC and the NNC provided above are somewhat different. This is due to the behavioral difference in the two control techniques. Figure 38 presents a contour plot, generated from the differences between the steering outputs of the hardware FLC and NNC blocks. As illustrated, the FLC and the NNC often provide different output signals. This difference increases as the back sensor distance measurement decreases, corresponding to an increase in sensor value. In some scenarios, the two controllers even provide completely opposite control outputs. As an example, with DSFS and DSBS equal to 400 and 500 respectively, the FLC suggests a steering output of 14 corresponding to the maximum steering angle of 30^{o} to the left while the NNC generates a steering output of 5 corresponding to 12^{o} to the right. For this particular case, in which both the front and the back sensor distance measurement is small in magnitude, the FLC provides the desired response. The undesired output of the NNC comes from the lack of training data covering the selected input ranges.

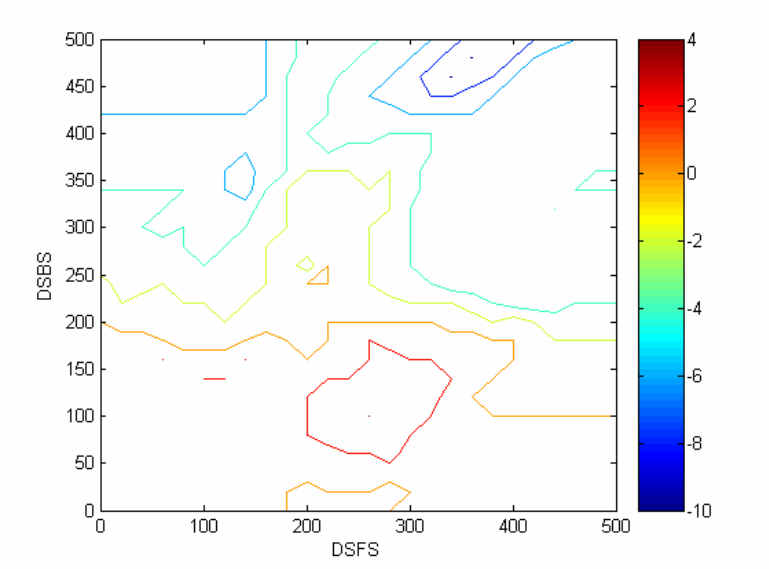


Figure 38: Contour Plot Summarizing the Differences between the NNC and the FLC Control Surfaces Generated by NIOS II

Another area of interest in the contour plot is when DSFS and DSBS are in the ranges of 200 to 300 and 50 to 100, respectively. As another example, let the side front and the side back sensor values be 250 and 100, respectively, corresponding to a smaller than desired front side sensor distance and a larger than desired back side sensor distance. In this particular case, the FLC and the NNC provide steering output values of 6 and 9, respectively. The steering output value of 9 provided by the NNC suggests a steering angle of 12^o to the left, which is intuitive for alignment of the vehicle to the wall. On the other hand, the steering output of 6 provided by the FLC corresponds to a steering angle of 5^o to the right, allowing the vehicle to move closer to the wall prior to aligning the vehicle.

Trajectory Designs

To evaluate and compare the performance of the FLC and the NNC, the prototype vehicle is tested in the real-world environment. Three trajectories are designed for this purpose. Different initial distances and wall-following distances are utilized and tested for each of the trajectories, such that an overall performance evaluation of the FLC and the NNC can be conducted. The three different trajectories designed for performance evaluation are the straight, angled and corner trajectories.

1) Straight Trajectory: The Straight Trajectory is utilized to evaluate the performance of the two controllers in maintaining a pre-defined distance. The physical dimensions of this trajectory are presented in Figure 39.

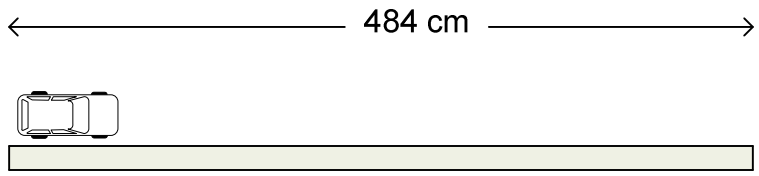


Figure 39: Straight Trajectory

2) Angled Trajectory: In the Angled Trajectory, two turns are utilized; a right-turn of 145 degrees and a left-turn of 140 degrees, as shown in Figure 40. The two turns are used to observe the performance of the two controllers when there is a change in the trajectory profile.

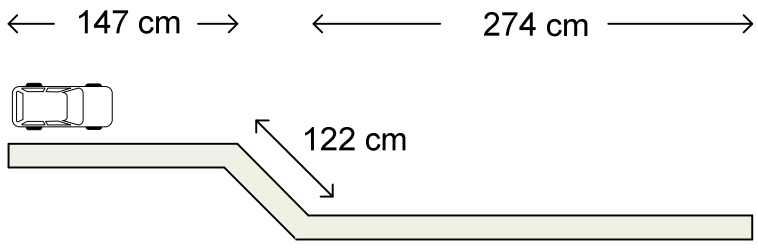


Figure 40: Angled Trajectory

3) Corner Trajectory: For the Corner Trajectory, the performance of the two controllers is compared when subjected to abrupt path changes. This trajectory involves a sharp 270-degree right-hand turn, as illustrated in Figure 41.

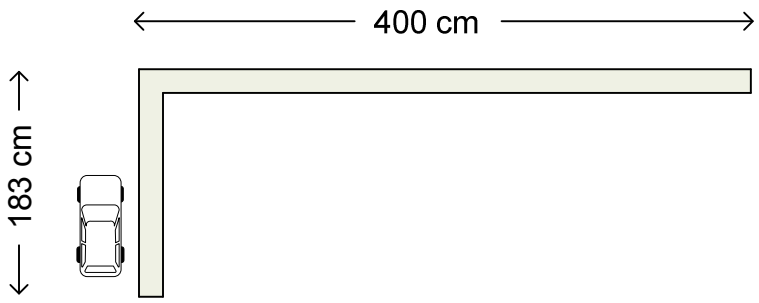


Figure 41: Corner Trajectory

Performance Measure

One method for controller performance evaluation is by means of an objective function, which quantizes the level of success in meeting the design goals. Generally, the function is designed such that either a maximum or minimum objective value is desired. For the FLC and the NNC, the goal is to generate an output such that the vehicle follows the wall at a predefined distance. In other words, both sensor measurements should maintain at a specified value. One objective function definition illustrating this goal is as follows:

$$TTE = f(DSFS, DSBS, DW) = \frac{MSE(DSFS - DW) + MSE(DSBS - DW)}{2} \quad (20)$$

In equation (20), *MSE*(*DSFS-DW*) and *MSE*(*DSBS-DW*) represent the mean squared errors of the differences between the front side sensor distance and the predefined distance and between the back side sensor distance and the predefined distance, respectively. The objective value, *TTE*, is also known as the trajectory tracking error. Upon closer examination of (20), for the scenario where the front side and back side sensor distances equal to the predefined distance, the function output is reduced to zero. Any variation between the predefined value and the sensor reading increases the function output value. Thus, a minimum objective value is desired.

Trajectory Tracking Error

For each trajectory, three different experiments were performed given different sets of initial distances (DI) and wall-following distances (DW) according to Table 12. The sensor readings and steering outputs from each of the experiments are gathered to graphically illustrate the vehicle paths. They are analyzed by computing the trajectory tracking errors as well as rise and fall times to evaluate

both the FLC and the NNC. It is important to note that the vehicle paths presented in this section represent typical results only. Although the vehicle paths may differ slightly when the same conditions and trajectories are repeated several times, the overall patterns remain the same.

Table 12: Experimental Initial and Wall-following Distances

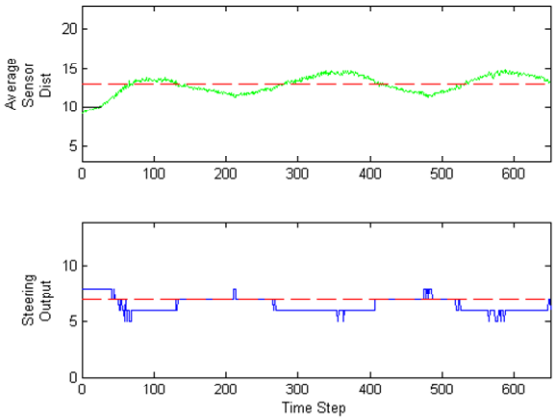
Experiment	DI (cm)	DW (cm)
#1	10	13
#2	20	13
#3	10	10

Straight Trajectory

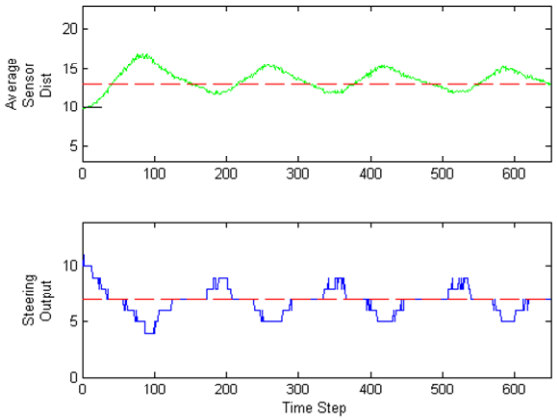
Figure 42 illustrates the average of the front and back sensor distances with the corresponding steering outputs for a straight path. The initial and wall-following distances from Table 12 are used to generate these results. The corresponding trajectory tracking errors as well as rise and fall times are summarized in Table 13. These values represent mean averages from at least three different test runs. In Table 13, the rise and fall times are defined as the time it takes for the front sensor measurement to reach the predefined distance.

Table 13: Straight Trajectory Tracking Error, Rise and Fall Times

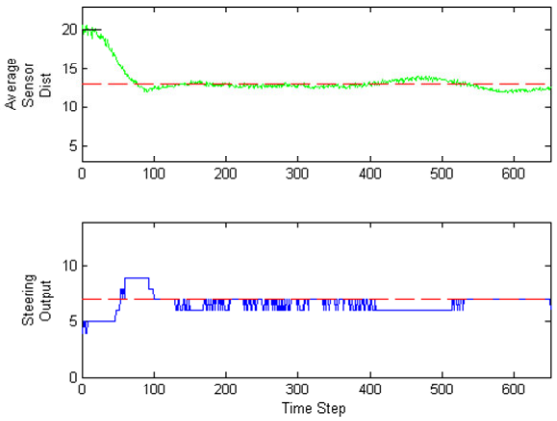
DI (cm)	DW (cm)	FLC TTE (cm^2)	FLC RISE/FALL TIMES (Time Step)	NNC TTE (cm^2)	NNC RISE/FALL TIMES (Time Step)
10	13	3.15	30	1.82	46
20	13	5.40	41	4.58	53
10	10	0.57	-	0.95	-



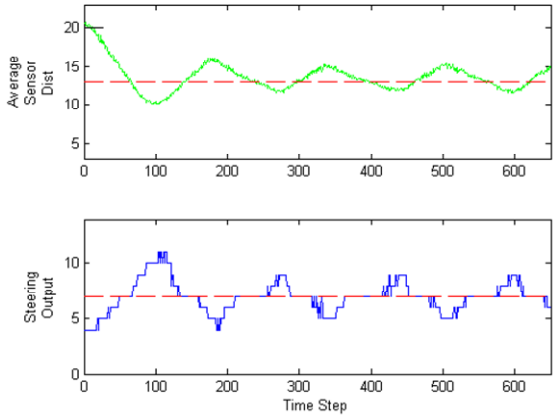
(a) NNC, DW = 13, DI = 10



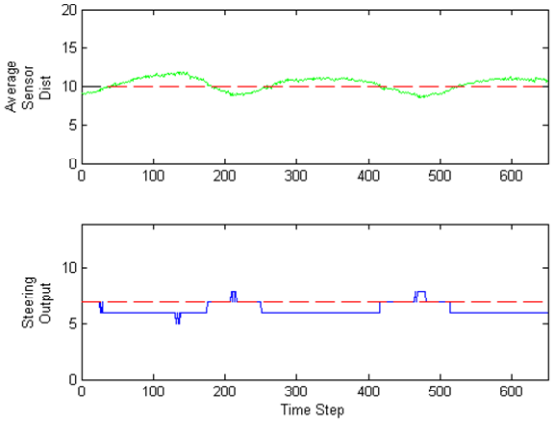
(b) FLC, DW = 13, DI = 10



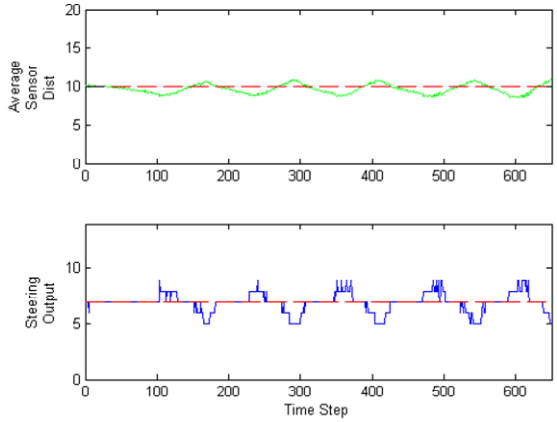
(c) NNC, DW = 13, DI = 20



(d) FLC, DW = 13, DI = 20



(e) NNC, DW = 10, DI = 10



(f) FLC, DW = 10, DI = 10

Figure 42: Sensor Distances and Steering Outputs of FLC and NNC for the Straight Trajectory

In following a predefined distance of 13 cm, the NNC requires longer rise and fall times compared to the FLC. The FLC approaches the desired distance within a shorter timeframe, but with the disadvantage of numerous oscillations experienced by the vehicle. Compared to the FLC, the NNC successfully maintains at the pre-defined distance without significant oscillations. The FLC generally utilizes bigger steering outputs, resulting in overshoots and in turn, oscillations. On the other hand, the NNC utilizes smaller steering outputs reducing oscillations, but requiring longer rise and fall times. In terms of trajectory tracking error, the NNC is superior to the FLC. In both scenarios, with initial distances of 10 cm and 20 cm, the trajectory tracking errors are slightly smaller for the NNC compared to the FLC. Similar to the first two scenarios, in following the trajectory at 10 cm, the vehicle oscillates more frequently when utilizing the FLC compared to the NNC. This can be attributed to more frequent adjustments in the steering outputs. In terms of trajectory tracking error, the FLC provides a smaller value compared to the NNC in direct contrast to the previously examined cases. For the straight trajectory, the FLC results in a larger trajectory error for the first two scenarios while a smaller error is resulted in the last case due to the differences in initial conditions. In the 10 cm case, the vehicle is launched at the predefined distance. It is unnecessary for the FLC to utilize large steering outputs, which would have resulted in large overshoots and trajectory errors as in the case for following the 13 cm distance. It can be seen that the NNC for 10 cm predefined distance provided similar performance to the case of the nominal 13 cm predefined distance. This demonstrates that the sensor mapping block can be easily applied to other wall-following distances in which training was not performed.

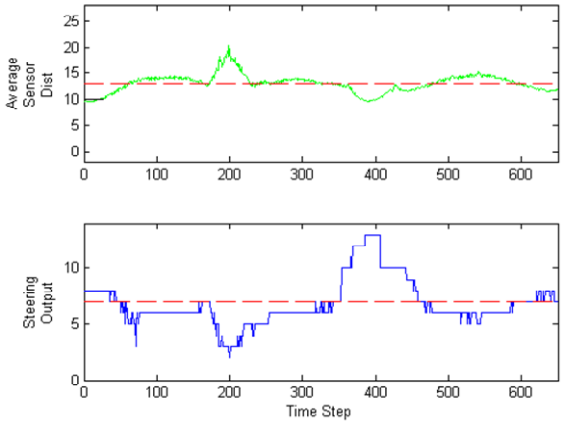
Angled Trajectory

Figure 43 illustrate the sensor distances and steering outputs of the FLC and the NNC as the vehicle traverses along the angled path with different initial conditions and wall-following distances. The corresponding trajectory tracking errors, rise and fall times for these scenarios are shown in Table 14.

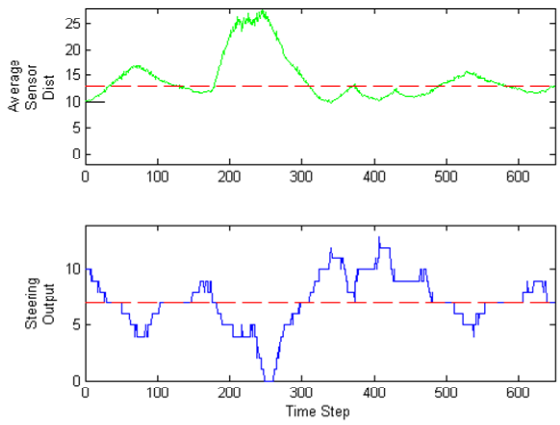
Table 14: Angled Trajectory Tracking Error, Rise and Fall Times

DI (cm)	DW (cm)	FLC TTE (cm^2)	FLC RISE/FALL TIMES (Time Step)	NNC TTE (cm^2)	NNC RISE/FALL TIMES (Time Step)
10	13	35.53	26	7.01	46
20	13	17.97	38	10.13	48
10	10	8.47	-	4.71	-

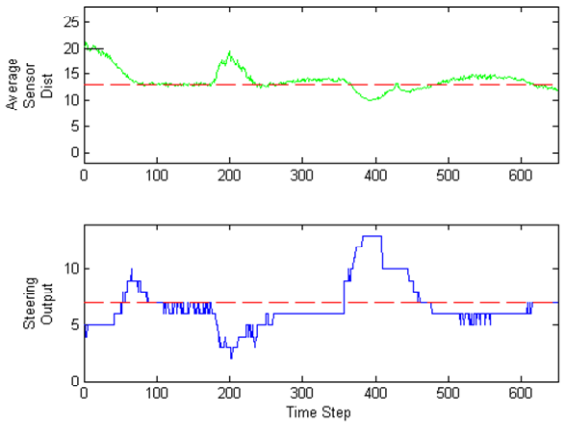
In following a predefined distance of 13 cm, it can be seen that the FLC approaches the desired distance faster compared to the NNC. Near the 200^{th} time step, in which the two controllers react to changes in the environment, the FLC generally results in higher average sensor distances. This is mainly due to the fact that at the time of the environmental changes, the FLC responds with smaller



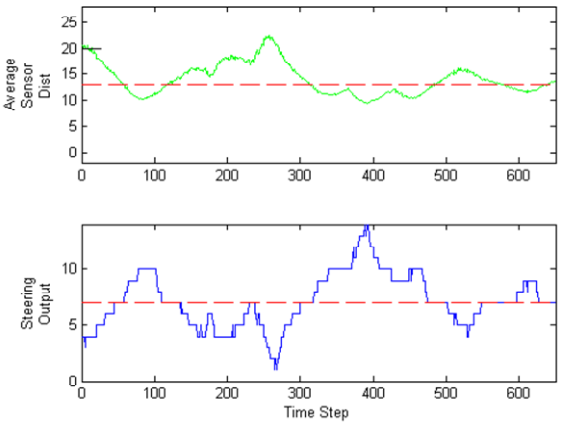
(a) NNC, DW = 13, DI = 10



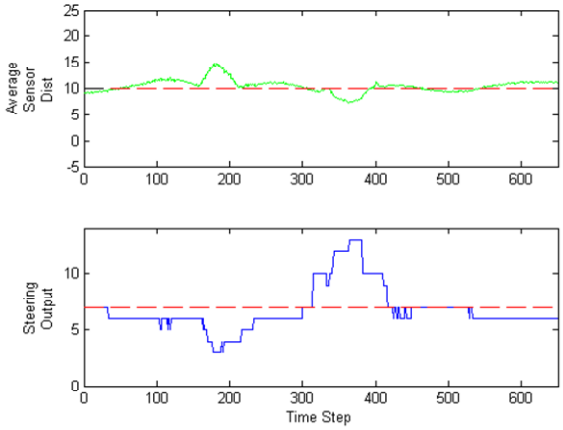
(b) FLC, DW = 13, DI = 10



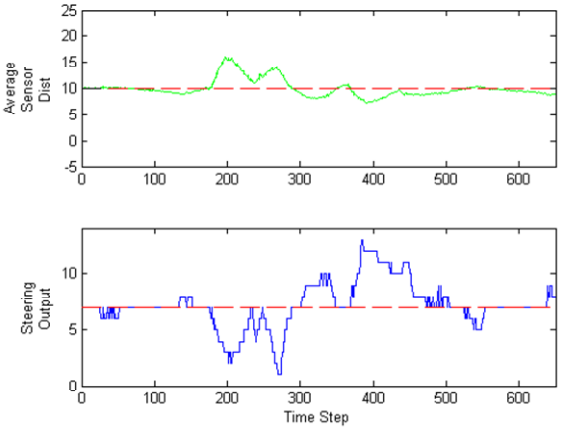
(c) NNC, DW = 13, DI = 20



(d) FLC, DW = 13, DI = 20



(e) NNC, DW = 10, DI = 10



(f) FLC, DW = 10, DI = 10

Figure 43: Sensor Distances and Steering Outputs of FLC and NNC for the Angled Trajectory

steering outputs, resulting in bigger sensor distances from the wall. For example, in time step 200 of Figure 43 (a)-(b), the FLC responds to an average sensor distance of approximately 23 cm with a steering output of 5, corresponding to 12^o to the right. In comparison, the NNC responds to an average distance of approximately 20 cm with a steering output of 2, corresponding to 26^o to the right. Although the FLC compensates with a large angle turn at a later point in time, the vehicle had already deviated from the intended path. Based on Table 14, the NNC provides superior performance in following the angled trajectory at 13 cm. With different initial distances, the trajectory errors are smaller for the NNC compared to the FLC. The difference is more significant in the case where the initial distance is 10 cm compared to the 20 cm case. An analysis on Figure 43 (b) and (d) shows that in the first scenario, the vehicle is performing left steering prior to the abrupt change, effectively making the change more significant. However, in the second scenario, the opposite is true. The vehicle is steering to the right in the direction of the change, effectively reducing its effect on the average sensor distance. Results from following a distance of 10 cm are consistent with that from the 13 cm scenarios. The trajectory error of the NNC is smaller than that of the FLC. This observation differs from the 10 cm scenario in which it was shown that the NNC had a greater trajectory error than the FLC when traversing the straight trajectory. For the straight trajectory, in which the FLC exerted small angle changes for path adjustments, this tactic allowed the vehicle to traverse close to the desired path. However, in the angled trajectory, the angle change required from the FLC was applied for an insufficient duration, creating a relatively large deviation from the intended path. This short duration can be partly attributed to the use of the following fuzzy rule:

R_1: IF DIS is NEG AND ANG is NEG THEN ϕ is c_1

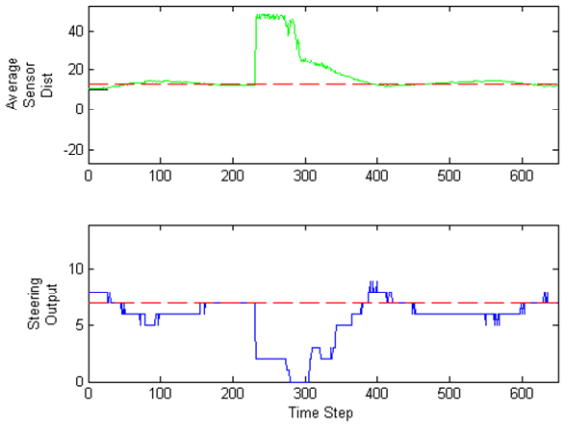
which states that if the back side sensor is close to the wall and the vehicle is oriented away from the wall, no steering is applied to the vehicle. In this scenario, the vehicle moves away from the wall. When the vehicle is at a sufficiently far distance from the wall, R_1 no longer applies and the vehicle is allowed to steer towards the right. This explains why the vehicle performs several short duration right turns in quick succession as seen in Figure 43 (f).

Corner Trajectory

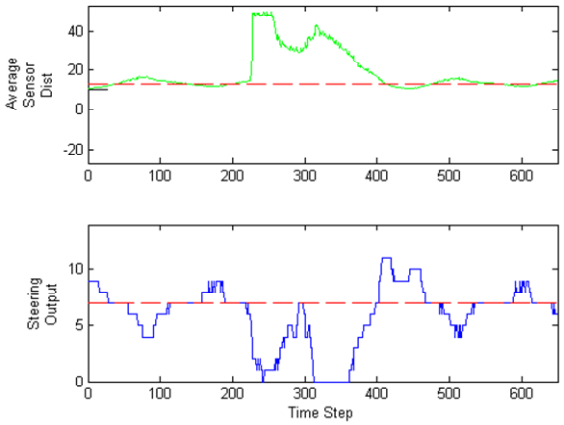
Figure 44 illustrates the average sensor distances and steering outputs of the FLC and the NNC as the robot performs the right-hand turn with various initial distances and a wall-following distance of 13 cm. Instead of two different wall-following distances, only one is tested since both the FLC and the NNC become ineffective in corner tracking with a smaller wall-following distance. Table 15 summarizes the trajectory tracking errors, rise and fall times for the two different scenarios. A close examination of Figure 44 shows that the average sensor distances of the FLC and the NNC peak at similar values. However, the average sensor distances in cases utilizing the NNC return close to the desired value faster than that of the FLC. Investigating the steering outputs of the FLC and the NNC shows that the NNC consistently applies a large steering angle for a longer duration when faced with abrupt changes. Based on Table 15, the FLC is inferior to the NNC in following the corner trajectory, as illustrated by higher trajectory tracking error values.

Table 15: Corner Trajectory Tracking Error, Rise and Fall Times

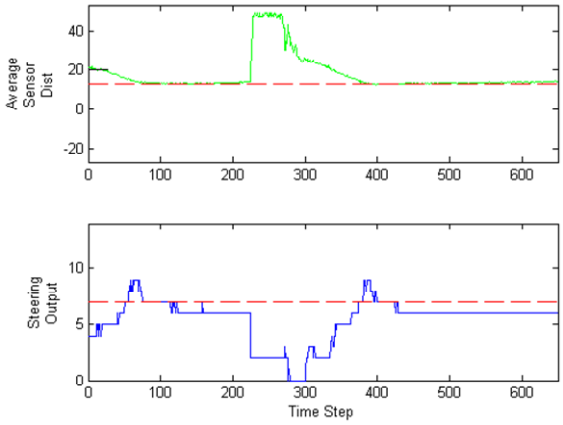
DI (cm)	DW (cm)	FLC TTE (cm^2)	FLC RISE/FALL TIMES (Time Step)	NNC TTE (cm^2)	NNC RISE/FALL TIMES (Time Step)
10	13	287.59	27	218.89	44
20	13	308.91	40	207.66	47



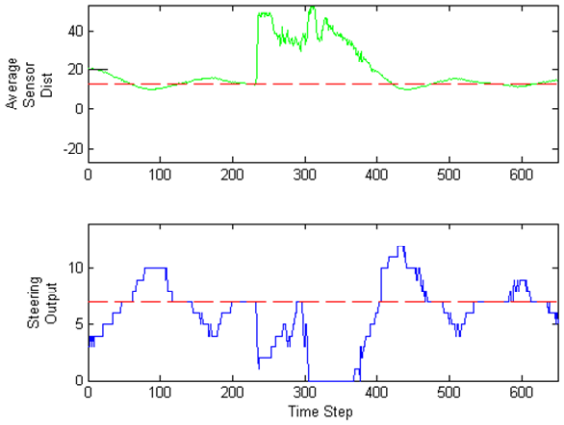
(a) NNC, DW = 13, DI = 10



(b) FLC, DW = 13, DI = 10



(c) NNC, DW = 13, DI = 20



(d) FLC, DW = 13, DI = 20

Figure 44: Sensor Distances and Steering Outputs of FLC and NNC for the Corner Trajectory

5.4 Summary

The design and implementation of two intelligent controllers, namely the FLC and the NNC, are presented. To evaluate their performance, three metrics are utilized. The first one is associated with the utilization efficiency of computational resources comparing the LE, memory and DSP requirements. In this respect, the FLC is superior with a hardware implementation that demands less LE, memory bits and DSP units. The second metric is the controller speed. The FLC and the NNC are compared in terms of maximum clock frequency, latency and throughput time. The NNC performs considerably better based on these factors. A higher maximum clock frequency is achieved along with smaller number of input-output cycles. Accordingly, a shorter throughput time is attained generating outputs at approximately four times the speed of the FLC, given that the clock frequency is set to its maximum value. The final metric is the experimental trajectory tracking error. Both controllers exhibit comparable performance in following all three trajectories. The robotic car with the FLC generally approaches the predefined distance faster resulting in shorter rise and fall times. However, the vehicle suffers from frequent undesirable oscillations as measured in the average sensor distance data. The NNC generally provides good tracking behavior with smaller trajectory tracking error, but requires longer rise and fall times for all cases.

One of the drawbacks of utilizing the NNC is the overhead cost in collecting high quality training data that closely simulates the intended trajectory and parameters when the vehicle operates autonomously. The controller is unlikely to provide the same performance in scenarios that were not originally considered by the training set compared to those that were. When the vehicle is tested on the trained paths (that is, paths used during training data collection), good performance is obtained regardless of the initial condition. In these scenarios, the network exhibits its learning and generalization abilities. In fact, smaller trajectory tracking errors are achieved in most trained scenarios compared to the FLC. Overall, neural networks are outstanding interpolators but poor extrapolators, meaning that they work well in circumstances covered by the training set, but underachieve otherwise. Generally speaking, the FLC should be selected if the amount of computational resources is limited. However, the NNC should be considered in cases where bandwidth and trajectory tracking error are limiting factors. The FLC and the NNC perform well for different situations. Thus, a controller combining features of the two is highly attractive.

Chapter 6
Hybrid Controller Design and Parallel Parking Application

With the advantages and drawbacks of the fuzzy logic and neural network controllers identified, a hybrid controller combining the desired features of both control units is investigated in this chapter. By means of switching, the hybrid controller selects between the steering output signal of the fuzzy logic and the neural network controller when performing the wall-following task. The effectiveness of the fuzzy logic and the neural network controller in following different trajectories triggers interest in their potential applications. Strong correlations with the lane-following and parallel parking tasks make the wall-following controllers applicable in parallel parking systems. In this chapter, the application of the fuzzy logic and the neural network controller in the parallel parking task is studied.

6.1 Hybrid Controller (HC)

Comparison of the two intelligent wall-following controllers shows that the FLC is less expensive in terms of computational resources but provides lower bandwidth. Based on experimental data, the NNC generally provides smaller trajectory tracking errors, but requires longer rise and fall times. The FLC, on the other hand, results in shorter rise and fall times with the disadvantage of undesirable oscillations in average sensor distances. Both controllers demonstrate strengths and weaknesses in serving as a wall-follower. To attain their benefits, a hybrid controller integrates the advantages of both. One method of realizing this type of hybrid controller is by utilizing the means of switching, in which a switching controller selects one of several control units for output generation. The selection algorithm embedded in this unit is often dependent on the design goals.

6.1.1 HC Design

A switching controller is developed for integrating the FLC and the NNC into a hybrid controller. At each time step, the switching controller selects one of the two steering outputs from the FLC and the NNC as the dominant steering signal. Based on experimental results, two potential controller selection rule sets are generated. The first is based on the trajectory type. As the NNC provides better performance in the trained trajectories (straight, angled and corner), it should be selected in those scenarios. For trajectories not covered in the training set, the FLC is preferred. This set of rules provides any hybrid type controllers with the benefit of the NNC in terms of small trajectory tracking error and the advantage of the FLC in its environment robustness. Nevertheless, realization of such a switching controller poses a challenge. Without an on-board camera unit capable of collecting large amounts of information about the environmental settings ahead of the vehicle, characteristics of trajectories such as degree of straightness and rate of change of curvature are difficult to assess. The second rule set is based on the current vehicle configuration with respect to its surrounding

environment. As the employment of the FLC results in shorter rise and fall times, it should be utilized in the initial stages of the wall-following experiments. Once the vehicle has reached within close proximity of the desired distance, the NNC should be chosen as it results in better performance for maintaining a predefined distance. Not only does this set of rules attain the benefit of the FLC in minimizing the rise and fall times, it also incorporates the small trajectory tracking error feature of the NNC. In addition, this rule set can easily be implemented since the switching condition is only dependent on the side sensor distance readings. Because the vehicle is not installed with a vision system, the second rule set is chosen to be implemented on the vehicle.

6.1.2 HC Implementation

Figure 45 shows the switching controller along with its connections to other hardware modules. The input channel of the switching controller is linked to the central controller for sensory information collection. The corresponding controller selection signal is connected to a multiplexer, which selects either the FLC or the NNC as the dominant output to the steering mechanism.

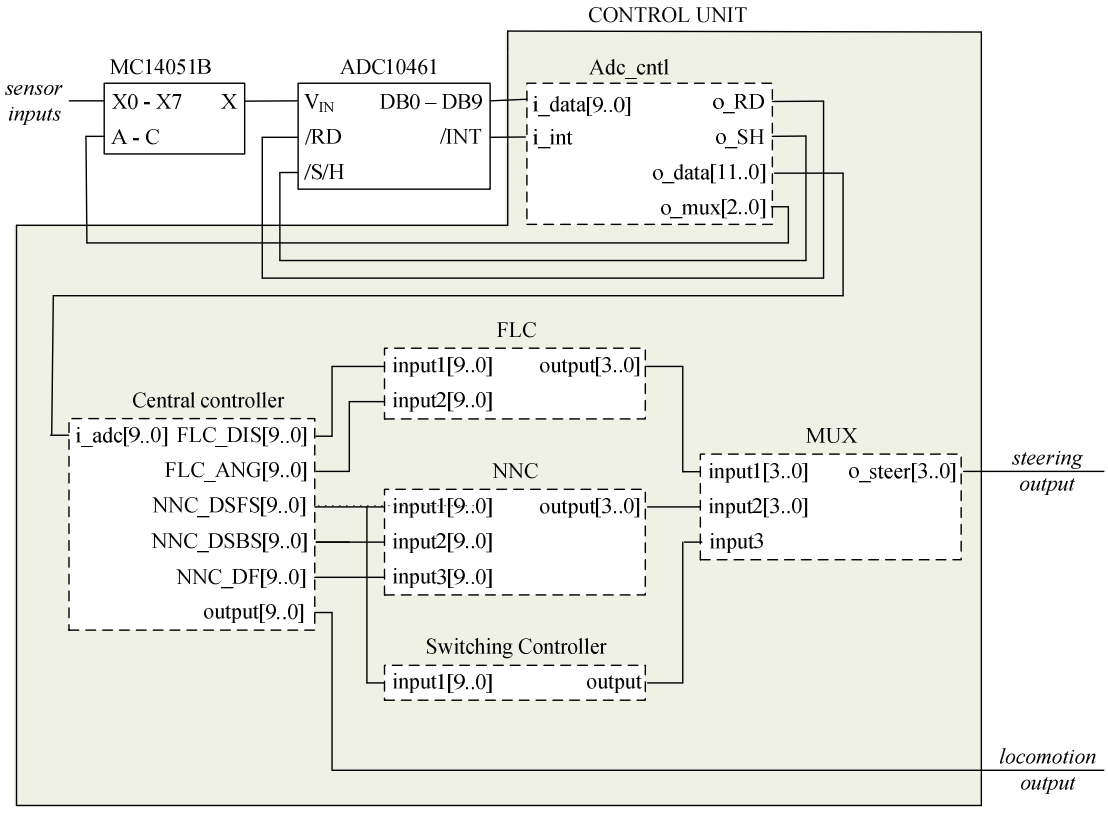


Figure 45: Hybrid Control Unit

6.1.3 Experimental Results

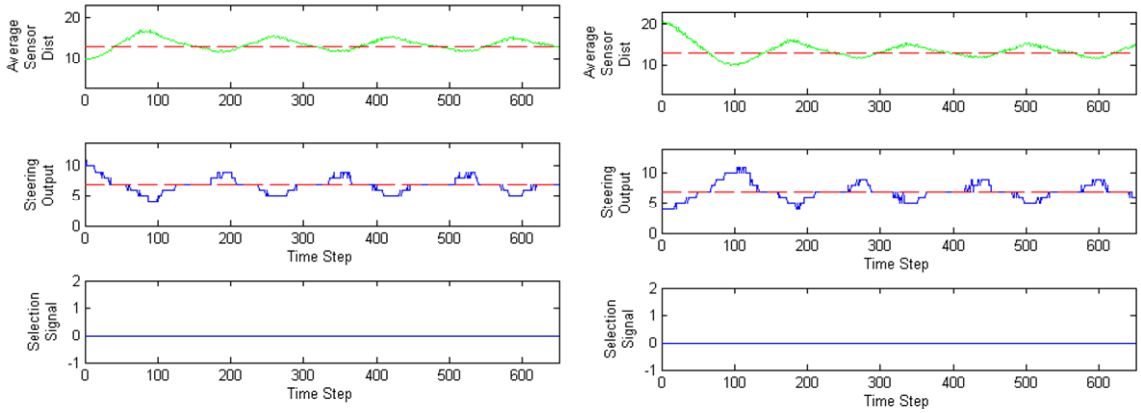
To demonstrate the feasibility of the hybrid controller, the prototype vehicle is tested along the same trajectories presented in Section 5.3.3. Figure 46 illustrates the average sensor distances and the steering outputs of the FLC, the NNC, and the HC as the robot traverses along the straight path with desired wall-following distance of 13 cm and initial distances of 10 cm and 20 cm. To closely examine the rise and fall times of each controller, Figure 47 shows only the first 80 time steps. For completeness, the controller selection decisions are also presented in both figures. They are assigned a value of 0 and 1 corresponding to the choice of the FLC and the NNC, respectively.

A close examination of Figure 46 shows that the NNC is chosen most of the time by the switching controller. The FLC is utilized only in the initial phase, accounting for approximately 3% of the entire trajectory. In general, the tracking performance of the hybrid controller is similar to that of the NNC, but with the advantage of reduced rise and fall times. Compared to the FLC, the hybrid controller has smaller trajectory tracking errors as illustrated in Table 16.

Table 16: Straight Trajectory Tracking Error, Rise and Fall Times for Different Controllers

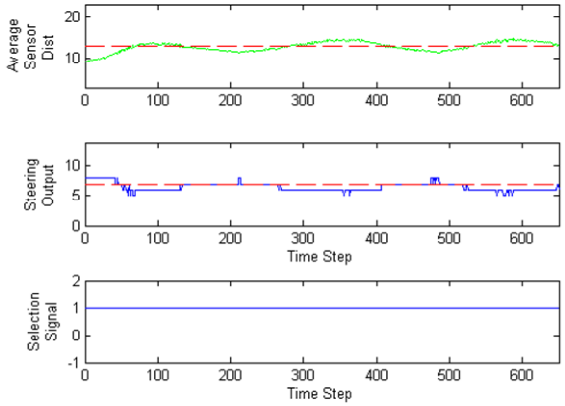
DI (cm)	DW (cm)	FLC TTE (cm^2)	FLC RISE/FALL TIMES (Time Step)	NNC TTE (cm^2)	NNC RISE/FALL TIMES (Time Step)	HC TTE (cm^2)	HC RISE/FALL TIMES (Time Step)
10	13	3.15	30	1.82	46	1.79	28
20	13	5.40	41	4.58	53	4.13	43

Table 17 summarizes the rise and fall times along with the trajectory tracking errors for different trajectory types and initial conditions. Consistent with observations from tracking the straight trajectory, the rise and fall times of the HC is smaller than that of the NNC. Furthermore, the HC produces trajectory tracking errors comparable to that of the NNC and smaller than that of the FLC.

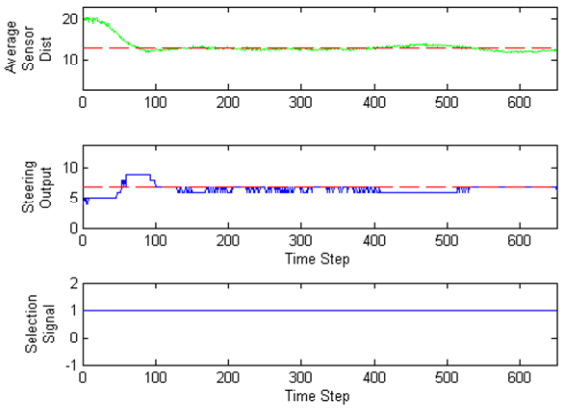


(a) FLC, DW = 13, DI = 10

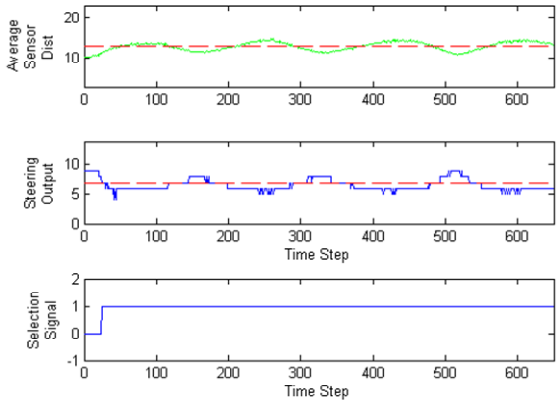
(b) FLC, DW = 13, DI = 20



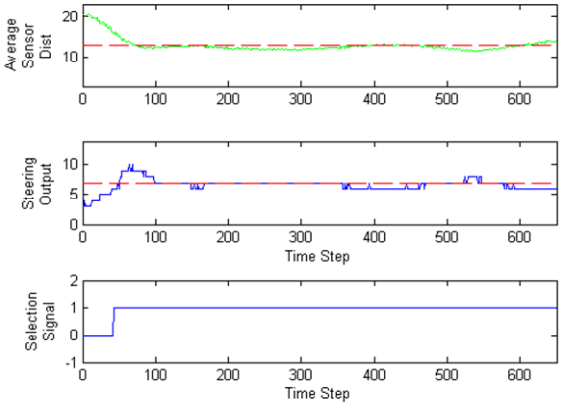
(c) NNC, DW = 13, DI = 10



(d) NNC, DW = 13, DI = 20

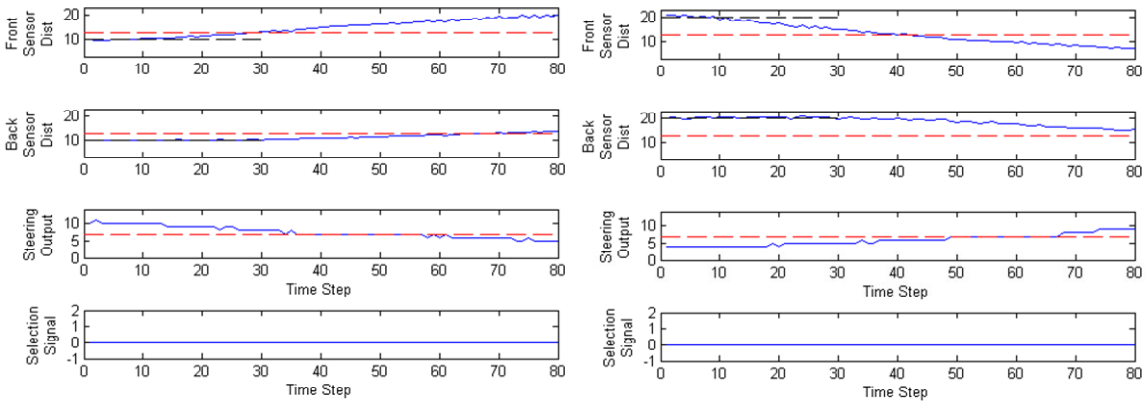


(e) HC, DW = 13, DI = 10



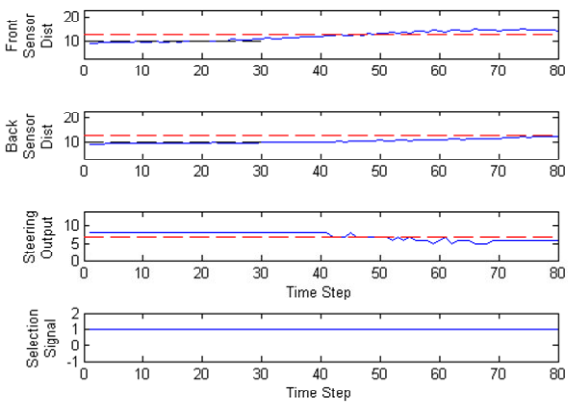
(f) HC, DW = 13, DI = 20

Figure 46: Sensor Distances and Steering Outputs of Different Controllers for the Straight Trajectory

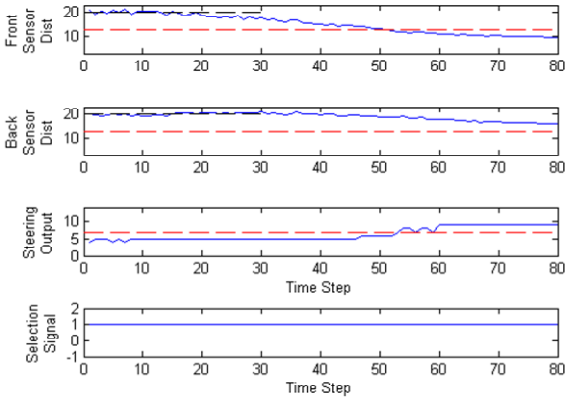


(a) FLC, DW = 13, DI = 10

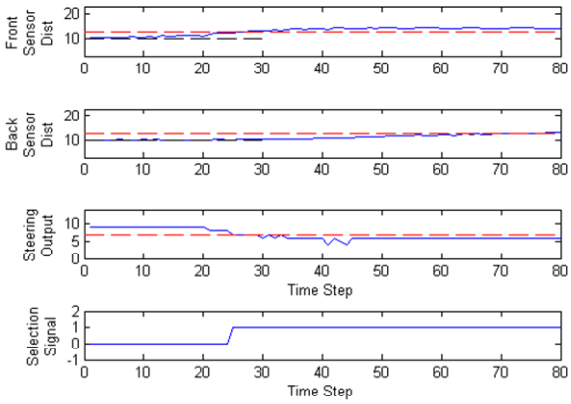
(b) FLC, DW = 13, DI = 20



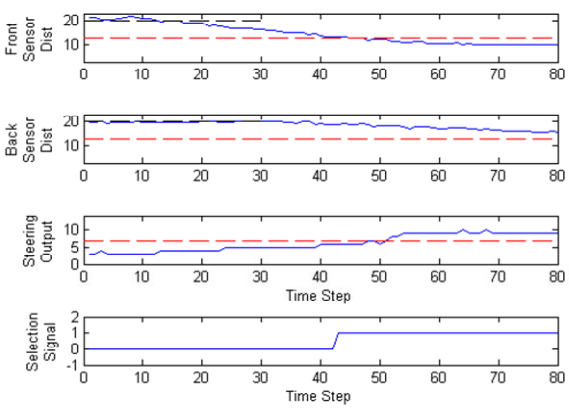
(c) NNC, DW = 13, DI = 10



(d) NNC, DW = 13, DI = 20



(e) HC, DW = 13, DI = 10



(f) HC, DW = 13, DI = 20

Figure 47: Sensor Distances and Steering Outputs of Different Controllers for the Straight Trajectory in Time Step 1-80

Table 17: Trajectory Tracking Error, Rise and Fall Times for Different Controllers and Trajectories

	DI (cm)	DW (cm)	FLC TTE (cm^2)	FLC RISE/FALL TIMES (Time Step)	NNC TTE (cm^2)	NNC RISE/FALL TIMES (Time Step)	HC TTE (cm^2)	HC RISE/FALL TIMES (Time Step)
ANGLED	10	13	35.53	26	7.01	46	6.47	28
ANGLED	20	13	17.97	38	10.13	48	9.39	40
CORNER	10	13	287.59	27	218.90	44	191.00	28
CORNER	20	13	308.91	40	207.66	47	201.20	39

The main advantage of the hybrid controller is the combination desirable qualities in both the FLC and the NNC. Small rise and fall times are inherited from the use of the FLC in the initial stage. At the same time, small trajectory tracking errors result from selection of the NNC once the vehicle is in close proximity to the predefined distance. However, these advantages come at the expense of an increase in computational resources as both the FLC and the NNC are implemented on the same hardware unit. Because the NNC is utilized most of the time, the same limitations of the NNC apply to the hybrid controller when in use.

6.2 Parallel Parking Application

A typical parallel parking procedure involves initial detection of a suitable parking space and ends when the car is parked successfully into the detected space [72]. To determine the current state in the parking cycle and the corresponding steering output, parking systems often make use of sensory data gathered by sensors equipped around the vehicle. This is very similar to wall-following systems, in which the steering outputs are determined by sensory information. Figure 48 illustrates the multiple steps involved in parallel parking a vehicle. First, the car moves parallel to the curb and then, stops shortly after it passes Car A. Next, it slowly reverses until its rear bumper aligns with the rear bumper of Car A. In the subsequent step, the driver of the car would turn the steering wheel all the way to the right and slowly move the vehicle backwards. As the front door passes the back bumper of Car A, the wheels are straightened. Once the front of the car is clear of Car A, the steering wheel is turned to the left as the vehicle continues reversing. Finally, the vehicle is moved back and forth to adjust the gaps in the front and rear of the vehicle.

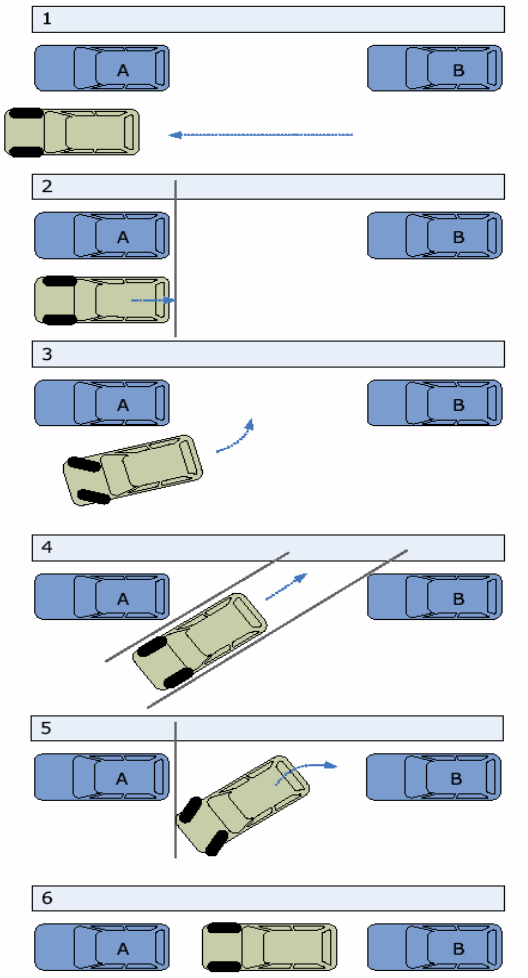
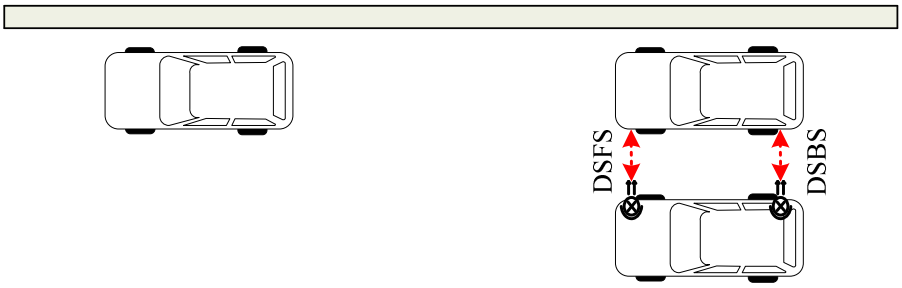


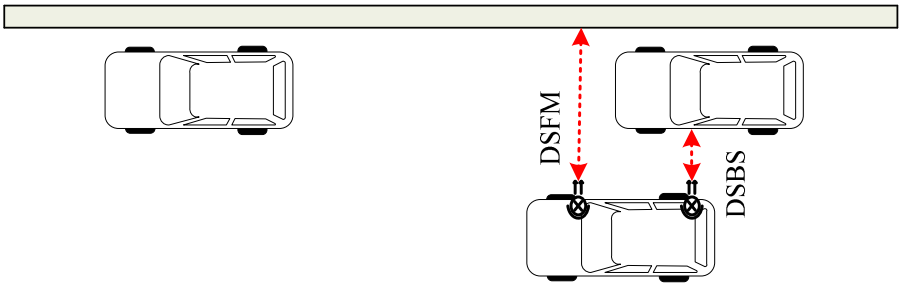
Figure 48: A Typical Parallel Parking Procedure [72]

6.2.1 Parallel Parking Controller Design

Realization of the parallel parking task requires all six sensors equipped on the vehicle. The parking controller uses the sensory information in performing one of five distinct states: Short Distance Wall-following, Long Distance Wall-following, Parking Maneuver, Reverse Adjustment and Forward Adjustment. In State 1, Short Distance Wall-following, the vehicle is in search of a parking space. The vehicle performs wall-following using the intelligent controllers discussed in previous chapters. When the vehicle detects the presence of other vehicles, the two short-range side sensor readings (DSFS and DSBS) in addition to the front sensor reading provide inputs to the wall-following controllers, as shown in Figure 49 (a). In the absence of a parked vehicle, the distance to the curb is measured. Due to the longer distances involved for curb distance measurements, which is out of the detection range of the short-range sensors, the short-range front side sensor reading (DSFS) is replaced with the mid-range front side sensor value (DSFM), as shown in Figure 49 (b).



(a) DSFS, DSBS

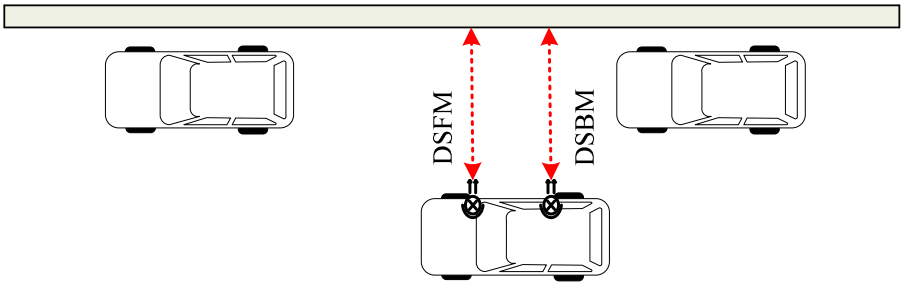


(b) DSFM, DSBS

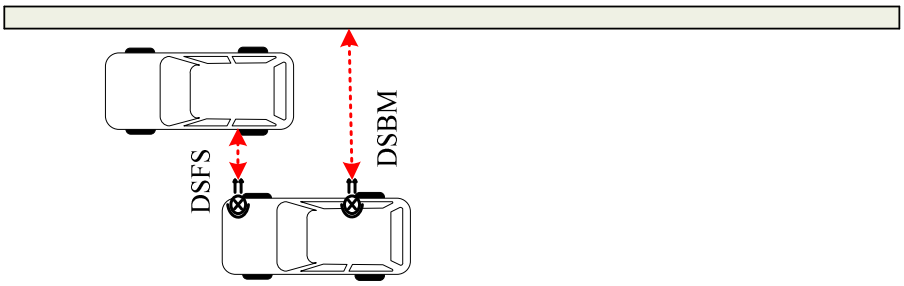
Figure 49: Short Distance Wall-following using Different Sensors

The transition from State 1 to State 2 is triggered by the absence of a parked car which signifies the presence of a potential parking space. In State 2, Long Distance Wall-following, the vehicle continues the parking space search by wall-following. The mid-range side sensor readings (DSFM and DSBM) along with the front sensor reading compose the input signals as the vehicle traverses along the potential spot, as shown in Figure 50 (a). Once the vehicle reaches the space in which the curb ends and the parked vehicle begins, the mid-range front side sensor reading (DSFM) is replaced with the short-range front side sensor value (DSFS), as shown in Figure 50 (b).

Transition from State 2 to subsequent states occurs immediately after the vehicle has traversed the entire potential space. Based on the time taken to traverse through the parking spot and the speed of the car, the size of the parking space is calculated. For a sufficiently large space, transition to State 3 is initiated. On the other hand, for a space of insufficient size, the controller defaults back to State 1. In State 3, Parking Maneuver, the vehicle attempts to move into the parking spot. The vehicle first reverses while using maximum right turn steering until it detects a certain distance on the rear sensor. It then initiates straight backward movement until the rear sensor measurement (DR) reaches a second distance value. Lastly, it performs left turn steering while continuously backing into the space.



(a) DSFM, DSBM



(b) DSFS, DSBM

Figure 50: Long Distance Wall-following using Different Sensors

Transition to State 4 is triggered once the vehicle has completely backed into the parking space. In State 4, Reverse Adjustment, the vehicle attempts to align itself with the wall. The two short-range side sensor readings (DSFS and DSBS) are used to perform wall-following in backward direction. In this state, the rear sensor reading (DR) is checked to prevent the vehicle from colliding with a parked ehicle behind it. State 4 terminates when the rear sensor value decreases to a predefined distance at which time transition to State 5 is triggered. In State 5, Forward Adjustment, the vehicle continues the wall-following maneuver, but in the forward direction using the two short-range side sensor readings (DSFS and DSBS). This state is completed when the front sensor value (DF) decreases to a predefined distance while the overall parking cycle can terminate once the two side sensor measurements are below a predefined value. If this criterion is not met, the controller transitions back to State 4. Alternating forward and backward wall-following movements continue until the final side sensor readings fall into satisfactory ranges.

6.2.2 Parallel Parking Controller Implementation

With five distinct states, the parallel parking controller can be realized by a finite state machine. Figure 51 presents the corresponding state transition diagram, in which state transitions are triggered by sensory information. Based on the current parking state and specified input conditions, the finite state machine generates input signals of the FLC or the NNC, which in turn determines the steering output. The best location to place the state machine is to embed it into the central controller. With

this design, all connections among various hardware modules previously discussed remain unchanged, as modifications are applied only to the internal design of the central controller. In other words, only VHDL coding describing the central controller behavior needs to be modified.

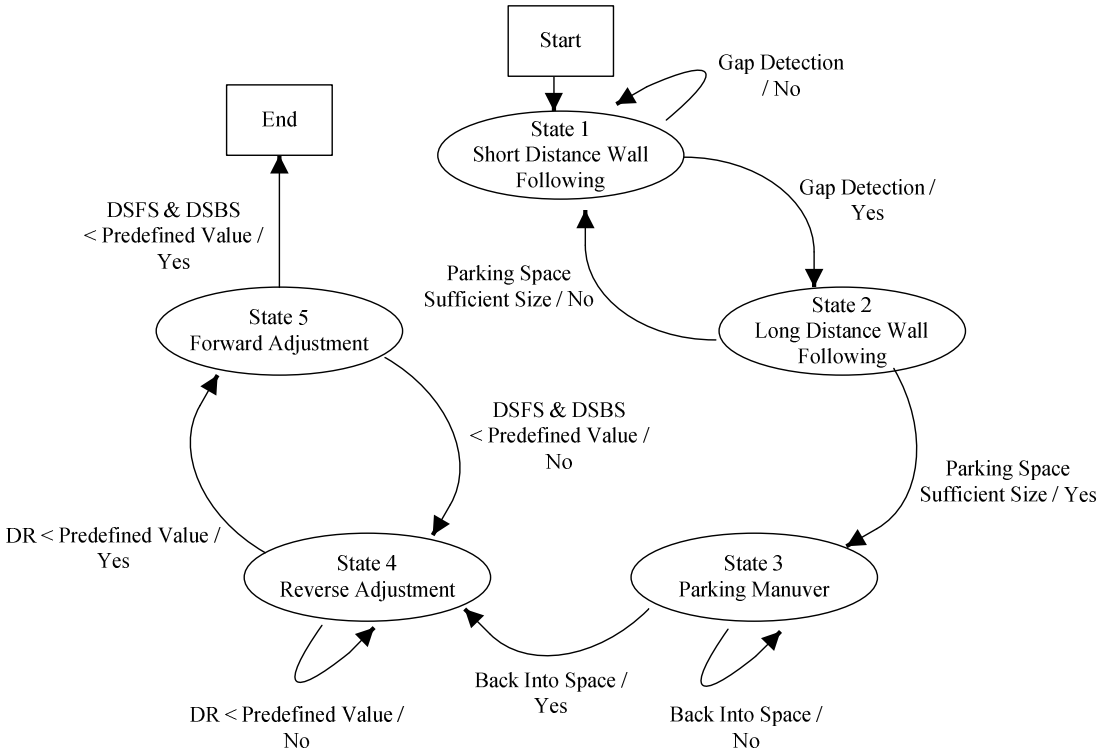


Figure 51: State Transition Diagram for Parallel Parking Controller

6.2.3 Experimental Results

To evaluate the performance of the parallel parking system, the prototype vehicle is tested on a setup that consists of a valid parking space along a straight wall. Figure 52 illustrates the physical dimensions of the environmental setting. Ideally, the prototype vehicle should first search for a valid parking space while performing the wall-following task. Once a parking spot is found, it should parallel park into the spot and adjust itself to be a predefined distance away from the wall.

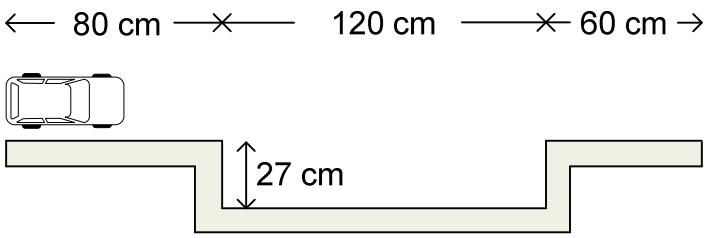
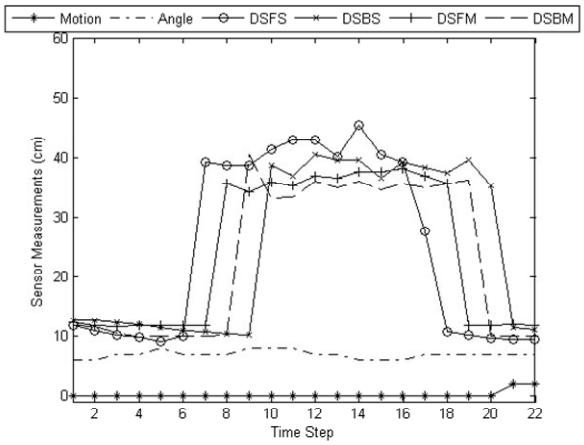
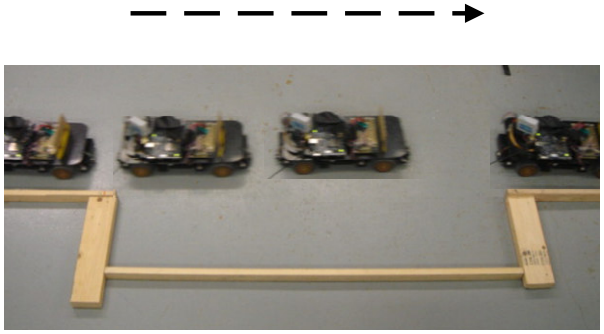


Figure 52: Straight Wall with a Valid Parking Space

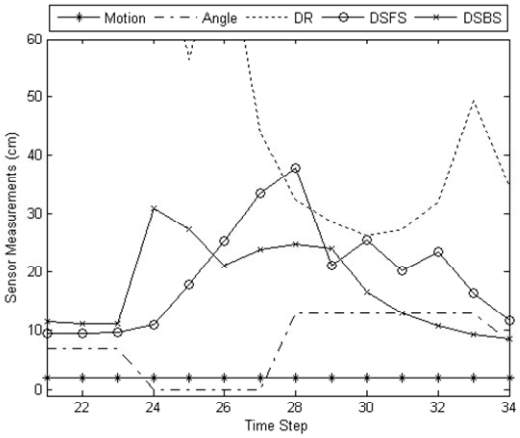
Figure 53 and Figure 54 log the sensory inputs and steering outputs of the parking system while utilizing the FLC and the NNC to perform the wall-following task. Corresponding camera snapshots for each maneuver are provided. As mentioned previously, the steering outputs are assigned a range of 0 to 6 for right turn steering, 7 for no steering, and 8 to 14 for left turn steering. For completeness, the motion outputs of the vehicle are also presented on the figures. They are assigned a value of 0, 1 or 2 corresponding to forward, stop and reverse motion, respectively. Figure 53 and Figure 54 are divided into three sections according to the objectives of the various states. State 1 and State 2 constitute the Gap Detection phase as the vehicle is in search for a parking space in these states. State 3 represents the Parking Maneuver phase, in which the parking maneuvers are executed to move the vehicle into the parking spot. The Final Adjustment phase is composed of State 4 and State 5. In these states, the vehicle performs forward and backward adjustments to position the vehicle in the desired final position.

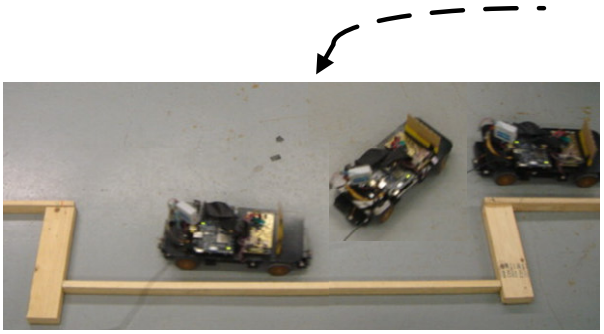
At the beginning of the parking cycle, the vehicle searches for a valid parking spot. A potential parking space is found when the side front sensor, DSFS, detects a distance longer than 22 cm. The potential parking space is then evaluated to determine its validity once the sensor reading returns to a distance shorter than 22 cm. In Figure 53 (a) and Figure 54 (a), DSFS is initially less than 20 cm. The value then increases to approximately 40 cm and returns to a value of less than 20 cm, indicating the vehicle has successfully detected a potential parking space. The controller then estimates the length of the parking space based on the speed of the vehicle and the length of time in which the detected distance is larger than 22 cm. In this scenario, since the parking space length is larger than the required size, the parking maneuver is initiated. Once a valid parking spot is found, the vehicle attempts to move into the space. According to Figure 53 (b) and Figure 54 (b), the front wheels are first steered to the maximum right, while the vehicle travels in the backward direction. They are then steered to the maximum left as the vehicle continuously moves backwards. As illustrated, the vehicle successfully moves into the parking space. The final step in the parking cycle is to adjust the vehicle to the desired location. In Figure 53 (c) and Figure 54 (c), the vehicle moves back and forth to adjust the front and rear gaps. Figure 53 and Figure 54 show that the vehicle successfully found the parking space and performed parallel parking into the given spot utilizing the FLC and the NNC. Hence, the designed parking system performs well in general.



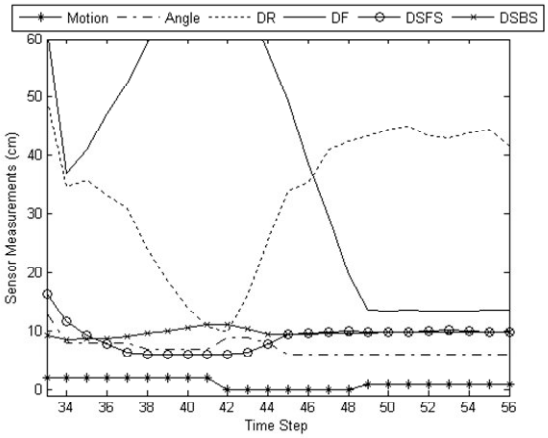


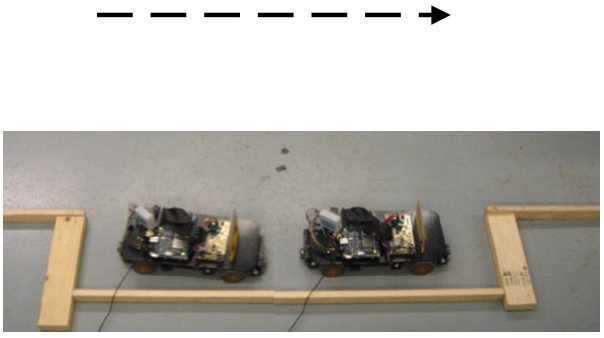
(a) NNC, Gap Detection





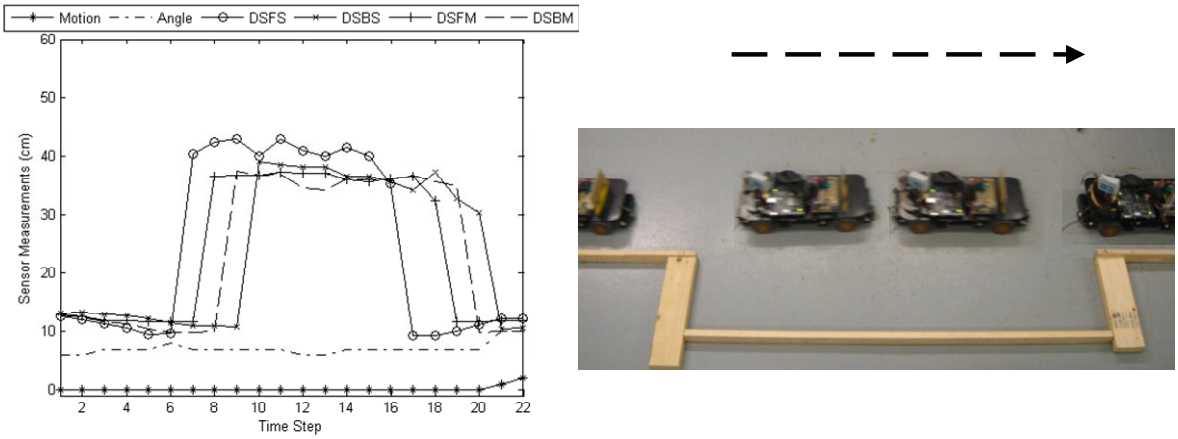
(b) NNC, Parking Maneuver



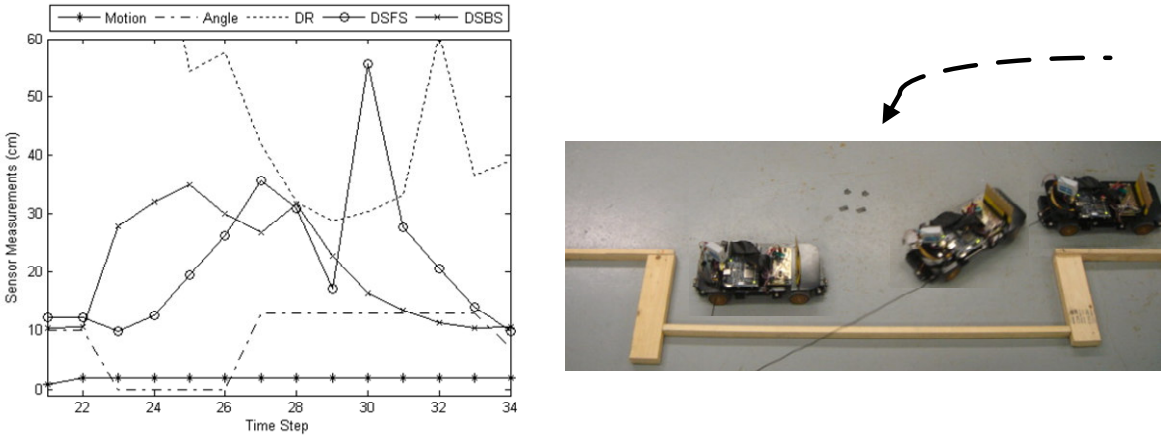


(c) NNC, Final Adjustments

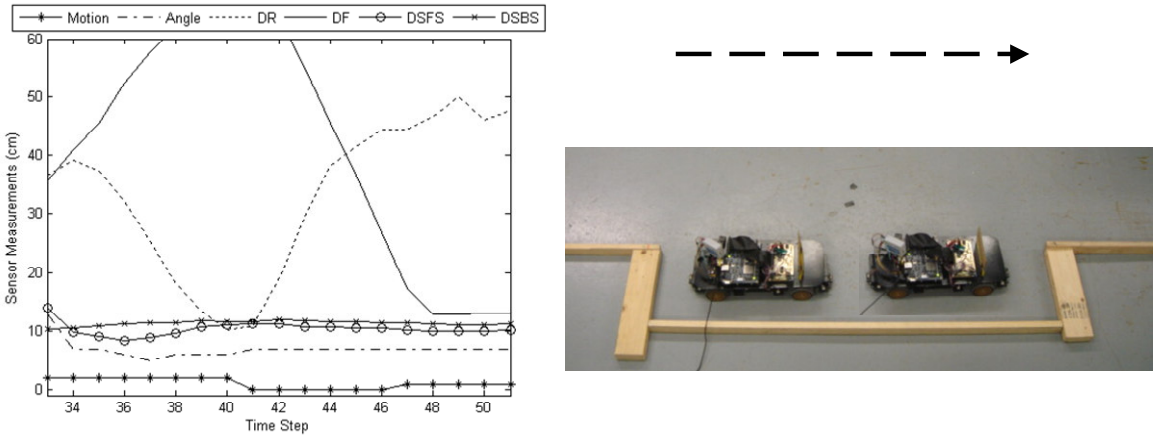
Figure 53: NNC Parallel Parking Results



(a) FLC, Gap Detection



(b) FLC, Parking Maneuver



(c) FLC, Final Adjustments

Figure 54: FLC Parallel Parking Results

6.3 Summary

This section provides the design and implementation of a hybrid controller, combining features of the fuzzy logic controller in terms of short rise and fall times and the neural network controller in terms of small trajectory tracking error. Experimental results show that the hybrid technique provides trajectory tracking error comparable to that of the NNC and better than that of the FLC. Also, the rise and fall times are comparable to the FLC and better than that of the NNC. The practical applicability of the FLC and the NNC in parking systems is demonstrated by creating a parallel parking control unit with the two wall-following controllers and a finite state machine. Based on experimental data, a vehicle incorporated with this system successfully found a valid parking space and performed parallel parking into this given spot.

Chapter 7
Conclusions

This research work focuses on the development and comparison of two soft computing based controllers, implemented on a reconfigurable hardware unit. The two control units are designed for realization of the wall-following task in vehicular systems, in which the goal is to control the steering angle of an autonomous vehicle based only on perception information from various sensors. A prototype vehicle is developed for testing and evaluating the two control algorithms. In general, the FLC utilizes less computational resources due to the use of integer representation and replacement of multiplication units with bit shifting circuitry. In comparison, the NNC utilizes fixed point signal representation and LUTs to implement the activation function and sensor value mapping, which is more costly in terms of hardware computational resources. Out of all available units, approximately 20% more LE, 1% more memory bits and 4% more DSP units are required in the implementation of the NNC compared to that of the FLC. Despite the greater hardware processing and storage element requirement, the NNC offers an advantage in bandwidth due to the parallel nature of neural networks along with the small number of hidden layers and neurons chosen. Only five clock cycles is required for propagation of a signal from input to output channels. This is approximately 37% reduction compared to the eight clock cycles required by the FLC. In addition, the maximum frequency of the NNC is 45.47 MHz, allowing it to support clock signals with frequencies two times faster than that supported by the FLC. Overall, the throughput time of the NNC is small, with a value of 109.96 ns, and approximately four times smaller than that of the FLC.

When comparing the control surfaces of the FLC and the NNC, a scenario in which the NNC is inferior to the FLC is discovered. In this particular case, the FLC provides the desired steering output to steer the vehicle away from the wall when the vehicle is initially in close proximity. However, the NNC suggests the opposite action shortening the already small distance between the vehicle and the wall. This result can be attributed to the lack of representative training patterns in the selected operating range since the quality of the neural network control unit is limited by the characteristics of its training set. In neural network controller design, the overhead cost in the collection of a set of good quality training data can be expensive. This is a significant drawback of neural networks. The cost in the design phase of fuzzy logic controllers is significantly lower since the data collection process is not required. The major cost in fuzzy logic controller design is essentially the process of translating expert knowledge into a set of fuzzy rules. Also, the definition of fuzzy variables and tuning of membership functions also pose a challenge. Experimental results show that the FLC generates shorter rise and fall times in all three predefined trajectories, at the expense of undesirable oscillations in average sensor distances. On the other hand, the NNC provides smaller trajectory tracking error in most scenarios, but with longer rise and fall times. Both controllers demonstrate strengths and weaknesses in trajectory following. The development of a hybrid controller in attaining the benefits of both controllers is shown to be overall successful. The hybrid controller selects the FLC output in the initial phase of the trajectory and later switches to the NNC when the desired

distance is reached. This takes advantage of the small rise and fall times of the FLC as well as the small trajectory error of the NNC. In the hybrid controller design, the main drawbacks of the NNC remains as it is selected for steering output in most scenarios. Also, the amount of resources is increased with both the FLC and the NNC implemented in the same hardware unit. The application of the fuzzy logic and the neural network controllers to the parallel parking task is deemed successful. By supplying the appropriate steering angles, both controllers parked the prototype vehicle into a valid parking space.

With the benefits of the NNC and the FLC identified, future work will focus on other methodologies for integration. For instance, by incorporating a vision sensor, the switching controller can select either the FLC or the NNC based on the trajectory type. Another example is the use of a neuro-fuzzy controller, in which the benefits of computational resource requirement of the FLC can be integrated with the small trajectory tracking error feature of the NNC without implementing two complete controllers.

Bibliography

[1] General Electric Company, "New GE Fuzzy Logic Front-Loading Washing Machine Cares for Wash INTELLI–GENTLY," April 2001. [Online]. Available: http://www.geconsumerproducts.com/pressroom/press_releases/appliances/worldwide/fzzyfrntld.htm

[2] Zojirushi America Corporation, "Neuro Fuzzy Rice Cooker & Warmer," April 2007. [Online]. Available: http://www.zojirushi.com/ourproducts/ricecookers/ns_zcc.html

[3] K. Balint, "Fair Isaac Corp. products protect consumers, companies," February 2005. [Online]. Available: http://www.signonsandiego.com/news/business/20050218-9999-1b18fair.html

[4] J. Nijhuis, S. Neusser, L. Spaanenburg, J. Heller, J. Sponnemann, "Evaluation of fuzzy and neural vehicle control," *Proceedings of 1992 IEEE International Conference on Computer Systems and Software Engineering*, May 1992, pp. 447-452.

[5] J. Godjevac, "Comparative study of fuzzy control, neural network control and neuro-fuzzy control," *Fuzzy Set Theory and Advanced Mathematical Applications*, Kluwer Academic Publishers, June 1995, pp. 291-322.

[6] L. Almeida, A. Mota and P. Fonseca, "Comparing control strategies for autonomous line-tracking robots," *5th International Workshop on Advanced Motion Control*, July 1998, pp. 542-547.

[7] R. Murray and S. Sastry, "Steering nonholonomic systems using sinusoids," *Proceedings of the 29th IEEE Conference on Decision and Control*, vol. 4, December 1990, pp. 2097-2101.

[8] I. Paromtchik and C. Laugier, "Autonomous parallel parking of a nonholonomic vehicle," *Proceedings of the 1996 IEEE Intelligent Vehicles Symposium*, September 1996, pp. 13-18.

[9] Y. Kanayama, A. Nilipour and C. Lelm, "A locomotion control method for autonomous vehicles," *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, vol. 2, April 1988, pp. 1315-1317.

[10] M. Egerstedt, X. Hu and A. Stotsky, "Control of mobile platforms using a virtual vehicle approach," *IEEE Transactions on Automatic Control*, vol. 46, issue 11, November 2001, pp. 1777-1782.

[11] C. Jung and C. Kelber, "An improved line-parabolic model for lane-following and curve detection," *Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*, 2005, pp. 131-138.

[12] J. Yang and J. Kim, "Sliding mode control for trajectory tracking of nonholonomic wheeled mobile robot," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, June 1999, pp. 578-587.
[13] G. Antonelli, S. Chiaverini and G. Fusco, "A Fuzzy-Logic-Based Approach for Mobile Robot Path Tracking," *IEEE Transactions on Fuzzy Systems*, vol. 15, issue 2, April 2007, pp. 211-221.
[14] T. Li and S. Chang, "Autonomous fuzzy parking control of a car-like mobile robot," *IEEE Transactions on Systems, Man and Cybernetics*, part A, vol. 33, issue 4, July 2003, pp. 451-465.
[15] C. Chao, C. Ho, S. Lin, T. Li, "Omni-directional vision-based parallel-parking control design for car-like mobile robot," *IEEE International Conference on Mechatronics*, July 2005, pp. 562-567.
[16] N. Rahman and A. Jafri, "Two layered behaviour based navigation of a mobile robot in an unstructured environment using fuzzy logic," *Proceedings of the IEEE Symposium on Emerging Technologies*, September 2005, pp. 230-235.
[17] C. Chiu, K. Lian and P. Liu, "Fuzzy gain scheduling for parallel parking a car-like robot," *IEEE Transactions on Control Systems Technology*, vol. 13, issue 6, November 2005, pp. 1084-1092.
[18] B. Reddy, B. Kimiaghalam and A. Homaifar, "Reactive real time behavior for mobile robots in unknown environments," *2004 IEEE International Symposium on Industrial Electronics*, vol. 1, May 2004, pp. 693-697.
[19] I. Baturone, F. Moreno-Velo, S. Sanchez-Solano and A. Ollero, "Automatic design of fuzzy controllers for car-like autonomous robots," *IEEE Transactions on Fuzzy Systems*, vol. 12, issue 4, August 2004, pp. 447-465.
[20] S. Yang and M. Meng, "Real-time collision-free motion planning of a mobile robot using a Neural Dynamics-based approach," *IEEE Transactions on Neural Networks*, vol. 14, issue 6, November 2003, pp. 1541-1552.
[21] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The American Journal of Physiology*, vol. 117, pp. 500–544, 1952.
[22] E. Zalama, J. Gomez, M. Paul and J. Peran, "Adaptive behavior navigation of a mobile robot," *IEEE Transactions on Systems, Man and Cybernetics*, part A, vol. 32, issue 1, January 2002, pp. 160-169

[23] G. Cicirelli, T. D'Orazio and A. Distante, "Neural Q-learning control architectures for a wall-following behavior," *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, October 2003, pp. 680-685.

[24] M. Quoy, S. Moga and P. Gaussier, "Dynamical neural networks for planning and low-level robot control," *IEEE Transactions on Systems, Man, and Cyberneticcs*, part. A, vol. 33, no. 4, July 2003, pp. 523-532.

[25] D. Janglova, "Neural networks in mobile robot motion," *International Journal of Advanced Robotic Systems,* vol. 1, no. 1, March 2004, pp. 15-22.

[26] F. Large, S. Sekhavat, C. Laugier and E. Gauthier, "Towards robust sensor-based maneuvers for a car-like vehicle," *Proceedings of 2000 IEEE International Conference on Robotics and Automation*, vol. 4, April 2000, pp. 3765-3770.

[27] J. Li, J. Yi and D. Zhao, "On-line rule generation for robotic behavior controller based on a neural-fuzzy inference network," *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, vol. 1, August 2004, pp. 558-563.

[28] M. Khoshnejad and K. Demirli, "Autonomous parallel parking of a car-like mobile robot by a neuro-fuzzy behavior-based controller," *Annual Meeting of the North American Fuzzy Information Processing Society*, June 2005, pp. 814-819.

[29] M. Er and C. Deng, "Online tuning of fuzzy inference systems using dynamic fuzzy Q-learning," *IEEE Transactions on Systems, Man and Cybernetics*, Part B, vol. 34, issue 3, June 2004, pp. 1478-1489.

[30] R. Oentaryo and M. Pasquier, "Self-trained automated parking system," *8th International Conference on Control, Automation, Robotics and Vision*, vol. 2, December 2004, pp. 1005-1010.

[31] P. Liatsis and C. Kammerer, "Robust parameter estimation in lane-following using a committee of local expert networks," *4th EURASIP Conference on Video/Image Processing and Multimedia Communications*, July 2003, vol. 1, pp. 161- 168.

[32] M. Mucientes, D. Moreno, A. Bugarín and S. Barro, "Design of a fuzzy controller in mobile robotics using genetic algorithms," *Applied Soft Computing*, vol. 7, issue 2, March 2007, pp. 540-546.

[33] M. Ho, P. Chan, A. Rad and C. Mak, "Truck backing up neural network controller optimized by genetic algorithms," *The 2003 Congress on Evolutionary Computation*, vol. 2, December 2003, pp. 944-950.

[34] Y. Zhao E. Collins and D. Dunlap, "Design of genetic fuzzy parallel parking control systems," *Proceedings of the 2003 American Control Conference*, vol. 5, June 2003, pp. 4107-4112.
[35] M. Mucientes and J. Casillas, "Quick Design of Fuzzy Controllers With Good Interpretability in Mobile Robotics," *IEEE Transactions on Fuzzy Systems*, Accepted for future publication.
[36] V. Peri and D. Simon, "Fuzzy logic control for an autonomous robot," *2005 Annual Meeting of the North American Fuzzy Information Processing Society*, June 2005, pp. 337-342.
[37] A. Divelbiss and J. Wen, "Trajectory tracking control of a car-trailer system," *IEEE Transactions on Control Systems Technology*, vol. 5, no. 3, May 1997, pp. 269-278.
[38] K. Jiang, "A sensor guided parallel parking system for nonholonomic vehicles," *Proceeding of 2000 IEEE Conference on Intelligent Transportation Systems*, October 2000, pp. 270-275.
[39] D. Kim, "An implementation of fuzzy logic controller on the reconfigurable FPGA system," *IEEE Transactions on Industrial Electronics*, vol. 47, no. 3, June 2002, pp. 703-715.
[40] T. Li, S. Chang and Y. Chen, "Implementation of human-like driving skills by autonomous fuzzy behavior control on an FPGA-based car-like mobile robot," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 5, October 2003, pp. 867-880.
[41] S. Bellis, K. Razeeb, C. Saha, K. Delaney, C. O'Mathuna, A. Pounds-Cornish, G. De Souza, M. Colley, H. Hagras, G. Clarke, V. Callaghan, C. Argyropoulos, C. Karistianos and G. Nikiforidis, "FPGA implementation of spiking neural networks - an initial step towards building tangible collaborative autonomous agents," *Proceedings of 2004 IEEE International Conference on Field-Programmable Technology*, December 2004, pp. 449-452.
[42] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, 1965, pp. 338-353.
[43] F. Karray and C. De Silva, *Soft Computing and Intelligent Systems Design*. Addison Wesley, 2004, pp. 77-78.
[44] J. Jang, C. Sun and E. Mizutani, *Neuro-Fuzzy and Soft Computing*. Prentice Hall, 1997, pp. 64-65.
[45] E. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, issue 1, 1975, pp. 1-13.
[46] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, January 1985, pp. 116-132.
[47] A. Zalzala and A. Morris, *Neural Networks for Robotic Control*. Ellis Horwood, 1996, pp. 11-16.

[48] M. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, issue 6, November 1994, pp. 989-993.

[49] T. Kohonen, *Self-organization and Associate Memory*. Springer-Verlag, 1984.

[50] C. Watkins, "Learning from delayed rewards," PhD dissertation, Cambridge University, 1989.

[51] Altera Corporation, "Stratix Architecture," *Stratix Device Handbook*, vol. 1, July 2005. [Online]. Available: http://www.altera.com/literature/hb/stx/ch_2_vol_1.pdf

[52] P. Chan and S. Mourad, *Digital Design using Field Programmable Gate Arrays*. PTR Prentice Hall, 1994, pp. 2-33.

[53] J. Oldfield and R. Dorf, *Field Programmable Gate Arrays*. John Wiley and Sons Inc., 1995, pp. 1-21.

[54] Z. Navabi, Digital Design and Implementation with Field Programmable Devices. Kluwer Academic Publishers, 2004, pp. 3-16.

[55] R. Rajsuman, *System-on-a-Chip: Design and Test*. Advantest America R&D Center Inc., 2000, pp. 3-29, 105-122.

[56] Altera Corporation, "Nios II Processor," *Nios II Processor Reference Handbook*, March 2007. [Online]. Available: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1_01.pdf

[57] Altera Corporation, "FFT MegaCore Function User Guide," April 2007. [Online]. Available: http://www.altera.com/literature/ug/ug_fft.pdf

[58] F. Alonso, "Experimental analysis of vehicle dynamics," April 2007. [Online]. Available: http://www.ectri.org/liens/yrs05/Session%206bis/Jimenez.pdf

[59] The MathWorks Inc., "Longitudinal Vehicle Dynamics," April 2007. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/physmod/drive/index.html?/access/helpdesk/help/toolbox/physmod/drive/longitudinalvehicledynamics.html&http://www.mathworks.com/cgi-bin/texis/webinator/search/?db=MSS&prox=page&rorder=750&rprox=750&rdfreq=500&rwfreq=500&rlead=250&sufs=0&order=r&is_summary_on=1&ResultCount=10&query=Longitudinal+Vehicle+Dynamics

[60] Sharp Corporation, "GP2D120 General Purpose Type Distance Measuring Sensors," April 2007. [Online]. Available: http://www.robotshop.ca/PDF/rbsha03_manual.pdf

[61] Sharp Corporation, "GP2D12/GP2D15 General Purpose Type Distance Measuring Sensors," April 2007. [Online]. Available: http://www.robotshop.ca/PDF/rbsha01_manual.pdf

[62] National Semiconductor Corporation, "ADC10461/ADC10462/ADC10464 10-Bit 600 ns A/D Converter with Input Multiplexer and Sample/Hold," April 2007. [Online]. Available: http://www.national.com/ds/DC/ADC10461.pdf

[63] On Semiconductor, "MC14051B, MC14052B, MC14053B Analog Multiplexers/Demultiplexers," April 2007. [Online]. Available: http://www.onsemi.com/pub/Collateral/MC14051B-D.PDF

[64] Altera Corporation, "Introduction," *Stratix Device Handbook*, vol. 1, July 2005. [Online]. Available: http://altera.com/literature/hb/stx/ch_1_vol_1.pdf

[65] T. Braunl, *Embedded Robotics*. Springer-Verlag, 2003, pp. 91-92.

[66] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000, pp. 15.

[67] M. Negnevitsky, *Artificial Intelligence*. Addison Wesley Limited, 2002, pp. 163-186.

[68] J. Bromley, "Synthesisable VHDL Fixed point Arithmetic Package," April 2007. [Online]. Available: http://www.doulos.com/knowhow/vhdl_designers_guide/models/fp_arith/

[69] M. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proceedings - Computers & Digital Techniques*, vol. 150, issue 6, November 2003, pp. 403-411.

[70] S. Merchant, G. Peterson and S. Kong, "Intrinsic embedded hardware evolution of block-based neural networks," *The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing*, June 2006, pp. 211-214.

[71] Altera Corporation, *Quartus II Development Software Handbook*, April 2007. [Online]. Available: http://www.altera.com/literature/lit-qts.jsp

[72] I. Song, F. Karray, Y. Dai, M. Masumodi and B. Ghaddar, "An intelligent car-like robot parking system design and implementation", *Third IEEE International Conference on Conference on Systems, Signals & Devices*, 2005.