National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

AUTONOMOUS VEHICLE DOCKING USING FUZZY LOGIC

by

Graham Eatherley

JANUARY 1994

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfilment of the
requirements for the degree of
Master of Applied Sciences
in Electrical Engineering

OTTAWA-CARLETON INSTITUTE FOR ELECTRICAL ENGINEERING

Department of Electrical Engineering
Faculty of Engineering

University of Ottawa
Ottawa, Ontario, CANADA

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Graham Eatherley

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by any other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Graham Eatherley

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF PHOTOGRAPHS

LIST OF TERMS

AI	artificial intelligence
A/D	analog to digital (as in *conversion from*)
DCL	differential competitive learning
EEPROM	electrically erasable programmable read only memory
HCMOS	high-density complementary metal-oxide semiconductor
I/O	input/output
KBG	knowledge base generator
LSB/MSB	least/most significant byte (of a two-byte word)
MCU	micro-controller unit
MESUR	Mars Environmental SURvey
MOSFET	metal-oxide semiconductor field effect transistor
NAFIPS	North American Fuzzy Information Processing Society
NASA	North American Space Agency
NN	neural network
PWM	pulse width modulation
RAM	random access memory
ROM	read only memory
RMS	remote manipulator system
SCI	serial communications interface
SPI	serial peripheral interface

ACKNOWLEDGEMENTS

I would like to thank Dr. Emil Petriu, my thesis supervisor, for his direction, guidance and advice. His interest in my work and his knowledge and experience have been of tremendous assistance in the research and preparation of this thesis.

Secondly, I would like to thank Larry Korba at the Autonomous Systems Laboratory of the National Research Council of Canada for his advice and assistance in developing the laser positioning system and for his helpful editorial comments.

In addition, I wish to thank both George Canetti and Vlad Angheleanu of Symbol Technologies, Canada for providing a miniature laser bar-code device used in the design of the laser-based positioning system.

Finally, I am grateful to my employer, Bell Canada, for giving me the opportunity to complete this research and for generously providing financial assistance while on leave of absence.

ABSTRACT

Autonomous control of robot vehicles has been studied recently by AI researchers with simulations and implementations emerging using both Neural Networks and Fuzzy Systems. The application is important primarily for autonomous control of planetary surface rovers and deep space vehicles where telepresence control (remote human operator) is precluded because of the long signal transit time.

The problem of vehicle docking is specifically addressed in this thesis and the results of experiments on a small model truck and trailer using a mechanical based position sensing systems is reported. Fuzzy logic controls both forward and backward motion of the vehicle to successfully dock the trailer.

Additionally, it is shown how the position sensing system can be improved using a small laser-based bar code engine. Future research including use of the system on other mobile robot platforms is discussed and we describe other research planned for the platform.

In this thesis the author shows that the use of fuzzy logic for vehicle control simplifies the analysis of motion required to produce proper docking. In particular, the discontinuity of control produced by switching from forward to reverse is readily handled by fuzzy logic.

CHAPTER 1

INTRODUCTION

1.1 Robotics

The field of Robotics is very old, stretching back even to the Egyptians and the Greeks, who incorporated movable arms into the statues of their Gods. In the eighteenth century, inventors produced lifelike *automata* in the shape of animals and humans which were capable of imitating lifelike behaviour. It was not until only recently that robots evolved from entertainment curiosities to productive, programmable machines useful in industry.

Although there is no accepted definition of the word *robot*, we can adopt a modern view of a robot as an intelligent connection of perception to action. This obviously excludes most of the robot machines currently used in industry since they do not perceive their environment at all, but merely respond to a programmed set of tasks. However, to be really useful, a robot must perceive and then act on that perception. If it can learn, then all the better; however, that is a more specific type of robot.

McKerrow, in his "Introduction to Robotics", [1] describes a *perception model* of robotics, wherein the physical world is transformed from an input state to an output state through the processes of measurement, modelling, perception, planning and action. In designing a system to achieve this transformation, we will normally work from the outputs, or actions, to the inputs, or measurements.

Artificial intelligence must have a central role in robotics if the connection of perception to action is to be intelligent [2]. Current industrial use of robotics employs very little intelligence. Much of the artificial intelligence research has been carried out in abstract domains and in computer simulations rather than in practical problems in the real physical world. Artificial intelligence typically is called upon to tackle the problems of how to represent knowledge (typically from sensor data), how to perceive crucial aspects of the environment important to the problem at hand, how to use knowledge in problem solving and how to act on that knowledge in the robot's current situation.

Current robotics research too often tries to copy human capabilities of mobility, dexterity, intelligence and sensory perception. Thus we have robot "legs, arms and hands", robot "brains" and robot "eyes and ears". Although nature is certainly an excellent role model, we have a long way to go before we can ever emulate even the simplest of life forms.

1.2 Applications of Robots

Robots have been used in diverse applications, from small turtle robots in classrooms, to the CanadaArm on the space shuttle. The introduction of robots into industry, in particular the automobile industry, has had a considerable impact on the manufacturing process. Most of the larger automobile manufacturers like General Motors and Honda, now have several totally automated assembly lines wherein entire cars are welded, assembled and painted with very little, if any, human intervention.

Another obvious application of robots is in the handling of dangerous materials, such as radioactive materials or corrosive chemicals, and in hostile environments, such as inside a reactor or under the sea. The laying of telecommunications cables under the seas and oceans is now assisted by remotely guided submersible robots. Remotely controlled submersibles have been in use for decades for exploration of the ocean floor; most notably was the use of an undersea robot to find, explore and film the wreck of the Titanic.

Robotics aids for the disabled range from prosthetic devices, such as replacement limbs or organs, to automatic wheelchairs and feeding machines for severely handicapped people. Because of the intimacy of these applications, the technological problems are very difficult. Sensor technology is pushed to its limit to protect the user and to perform the desired tasks, usually in an unstructured environment.

Space, the final frontier, and an area of intense AI and Robotics research. Manned long range exploratory missions are currently not being planned by world space agencies because of the high cost and dangers associated with these missions. To date, surface exploration of our nearest neighbours, Mars and Venus, has been limited to single, larger vehicles having only limited movement and exploratory capabilities.

1.3 NASA Planetary Rover Requirements

To date, the NASA program has addressed a variety of vehicle sizes and types, including those whose weight varies from a few kilograms to several thousand kilograms. Most of these vehicles were larger testbeds demonstrating autonomous navigation.

Due to the problems of advocating a large, costly vehicle with drastic potential for single point failure, NASA planners have changed their focus to the development and deployment of several smaller rovers for exploratory missions [3].

The next mission to Mars currently under study is the Mars Environmental Survey (MESUR), with four launches planned over a five year period (1999-2003). It is likely that this mission will consider significant use of micro-rovers, operating somewhat autonomously under the control of a central control station on the planet's surface.

The research reported in this thesis was inspired by these goals and the need to demonstrate autonomous vehicle behaviours in a real-world situation. The Canadian Space Agency is interested in being a part of the Mars missions currently planned by NASA and to that end has been developing and testing smaller wheeled rovers capable of autonomous navigation and exploration. It is hoped that the research begun here can be extended and applied to these vehicles.

CHAPTER 2

FUZZY LOGIC OVERVIEW

2.1 Fuzzy Origins

Fuzzy logic theory was developed by Professor Lofti Zadeh at the University of California at Berkeley in 1965. In his seminal paper "Fuzzy Sets" [4], Zadeh formally developed multi-valued set theory, introducing the term *fuzzy* into the technical literature. Fuzzy logic is a superset of conventional (*Boolean*) logic that has been extended to handle the concept of partial truth. Unlike conventional (discrete or *crisp*) set theory, where objects are either in or out of the set, Zadeh's theory allows objects to have partial membership.

Zadeh states that rather than regarding fuzzy theory as a single theory, we should regard the process of *fuzzification* as a methodology to generalize any specific theory from a crisp to a continuous form. Recently researchers have also introduced *fuzzy calculus, fuzzy differential equations,* and so on [5].

Just as there is a strong relationship between Boolean logic and the concept of a subset, there is a similar strong relationship between fuzzy logic and fuzzy subset theory. In classical set theory, a subset U of a set S can be defined as a mapping from the elements of S to the elements of the set {0,1},

$$U: S \rightarrow \{0, 1\}$$

This mapping may be represented as a set of ordered pairs, with exactly one ordered pair present for each element of S. The first element of the ordered pair is an element of the set S, and the second element is an element of the set {0, 1}. The value 0 is used to represent non-membership and the value 1 is used to represent membership.

The truth or falsity of the statement, x is in U, is then determined by finding the ordered pair whose first element is x. The statement is true if the second element of the ordered pair is 1 and the statement is false if it is 0.

2.2 Degrees of Membership

Similarly, a fuzzy subset *F* of a set *S* can be defined as a set of ordered pairs, each with the first element from *S*, and the second element from the interval [0, 1], with exactly one ordered pair present for each element of *S*. This defines a mapping between elements of the set *S* and values in the interval [0, 1]. The value 0 is now used to represent *complete* non-membership while the value 1 represents *complete* membership. Values in between are used to represent intermediate *degrees of membership*. The set *S* is referred to as the *universe of discourse* for the fuzzy subset *F*. Usually the mapping is described as a function, the *membership function* of *F*.

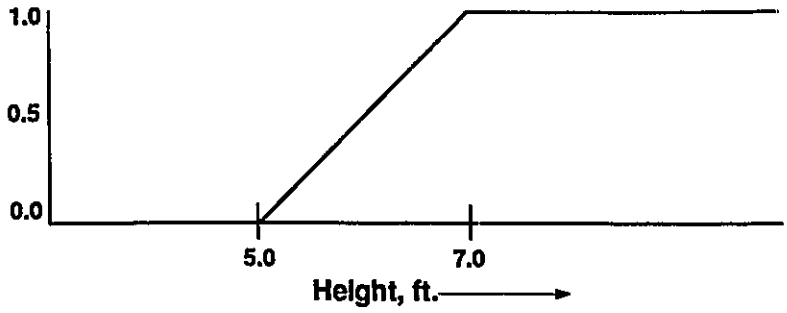
The degree to which the statement, *x* is in *F*, is true is determined by finding the ordered pair whose first element is *x*. The degree of truth of the statement is the second element of the ordered pair. In practice, the terms *membership function* and *fuzzy subset* are used interchangeably.

What we refer to as *fuzziness* takes in one aspect of uncertainty or ambiguity. Typically, this ambiguity is expressed in the definition of a concept or the meaning of a word (usually an adjective). If we say "old person", we cannot clearly determine who is old and who is not. The ambiguity of "tall person" resides in the adjective "tall".

As an example, we can define "tall" to be a *linguistic variable* representing our cognitive category of "tallness". To each person on the universe of discourse, we assign a degree of membership in the fuzzy subset "tall". We will do this based on their "height" (a measure of tallness):

$$\text{tall}(x) = \begin{cases} 0, & \text{if height}(x) <= 5 \text{ ft.} \\ (\text{height}(x) - 5)/2, & \text{if } 5 \text{ ft.} < \text{height}(x) < 7 \text{ ft.} \\ 1, & \text{if height}(x) >= 7 \text{ ft.} \end{cases}$$

A graph of this looks like:



Given this definition, some example values might be:

PERSON	HEIGHT	DEGREE OF TALLNESS
Billy	3 ft.	0.00
Judi	5 ft. 2 in.	0.08
Graham	5 ft. 11 in.	0.46
Mark	6 ft.	0.50
Kareem	7 ft. 2 in.	1.00

Expressions like "A is X" can be interpreted as degrees of truth, i.e., "Mark is TALL" = 0.5.

Note that membership functions are not usually as simple as the straight line function depicted in this example. As a minimum, they tend to be trapezoidal in shape, and they can be much more complex, for example quadratic or Gaussian in form. In addition, although the most common case is for a membership function to be based on only one variable, it is quite legitimate to base it on more than one. For example, the membership function for "tall" could depend on both height and age ("he's tall for his age").

2.3 Classical Relationships

Most of the definitions and operations associated with fuzzy sets are straight forward extensions of those from ordinary set theory, although there are several alternative definitions possible. Thus the notions of subset, union, intersection and properties of commutativity, associativity and distributivity are easily described for fuzzy sets. Union (logic OR) is usually taken as the *maximum* of candidate values while intersection (logic AND) is usually taken as the *minimum* of candidate values. In addition, most of the Boolean logic operations like AND, OR and NOT exist in fuzzy logic and produce the same truth tables as Boolean for values of 0 and 1. This is known as the *extension principle*, which states that the classical results of Boolean logic are recovered from fuzzy logic operations when all fuzzy membership grades are restricted to the traditional set {0, 1}. This effectively establishes fuzzy subsets and logic as a true generalization of classical set theory and logic. In fact, traditional or *crisp* subsets are, by this reasoning, fuzzy subsets of a special type and thus there is no conflict between fuzzy and crisp methods.

More information on fuzzy logic operators can be found from Bandler and Kohout [6] and from Dubois et al [7]. Valuable information can also be gleaned from Zadeh's many papers on fuzzy logic [8] [9] [10].

2.4 Fuzzy Control Theory

Modern control theory is exact, precise and logical, harbouring no hint of *fuzziness*. Today, however, the subjects of control have become increasingly larger in scale, requiring more advanced and complex control systems, like those used to control robots and space vehicles. A tremendous amount of computing power would be needed to execute such complicated control using modern theory. Fuzzy theory provides the means to overcome these limitations of control theory.

As a practical tool an important step occurred around 1970 with the work of Mamdani [11] on fuzzy or *approximate reasoning*. Approximate reasoning allows inferences to be made on the basis of input information through fuzzy sets via membership functions. This deals with the fundamental notion of *knowledge, production rules* or IF-THEN rules.

In fuzzy expert systems, rules are of the form:

IF distance is far AND direction is forward THEN speed is fast

Here *distance* and *direction* are input variables, *speed* is an output variable and *far, forward* and *fast* are fuzzy subsets or membership functions. The *antecedent* (the rule's premise) describes to what degree the rule applies, while the *consequent* (the rules conclusion) assigns a membership function to the output variable(s). The set of rules in a fuzzy expert system is referred to as the *rule base* or *knowledge base,* just as in a traditional expert system. The general inference process proceeds in three or four steps:

1. Under *fuzzification,* the membership functions defined on the input variables are applied to their actual values, to determine the degree of truth for each rule premise.

2. Under *inference,* the truth value for the premise of each rule is computed and applied to the conclusion part of each rule. This results in one fuzzy subset to be assigned to each output variable for each rule. Usually only MIN or PRODUCT are used as inference rules. In MIN inferencing, the output membership function is clipped at a height corresponding to the rule premise's computed degree of truth (fuzzy logic AND). In PRODUCT inferencing, the output membership function is scaled by the degree of truth.

3. Under *composition,* all of the fuzzy subsets assigned to each output variable are combined to form a single fuzzy subset for each output variable. In this case, usually MAX or SUM are used. In MAX composition, the combined output fuzzy subset is constructed by taking the point-wise maximum over all of the fuzzy subsets assigned to the output variable by the inference rule (fuzzy logic OR). In SUM composition, the output fuzzy subset is constructed by taking the point-wise sum over the fuzzy subsets. Note that this can result in truth values greater than 1. For this reason, SUM composition is usually followed by normalization.

4. Finally is the optional *defuzzification* which is used to convert the fuzzy output set to a crisp number. There are many defuzzification methods but the most common techniques employed are the *maximum* method and the *centroid* or *center of gravity* method. The simplest defuzzification method produces the point at which the possibility distribution of the control action reaches a maximum value; that is, we choose the element that has maximal membership in the output fuzzy set. This is sometimes referred to as the MAX *criterion* method or *maximum-membership defuzzification* scheme.

This simple defuzzification method has two fundamental problems associated with it. With correlation-minimum encoding, the additively combined output vector tends to be flat over much of the universe of discourse. For continuous membership functions this leads to infinitely many modes. In addition, this scheme ignores much of the information in the fuzzy output set. If the output set is highly asymmetric, many output distributions will share the same mode.

The more popular *centroidal defuzzification* technique uses more of the information in the fuzzy output set to compute the crisp value as the centroid or center of mass of the output set. If the fuzzy membership functions are trapezoidal then this fuzzy centroid can be calculated using:

$$z_0 = \frac{\sum_{i=1}^{n} \mu_i * x_i}{\sum_{i=1}^{n} \mu_i}$$

where x_i is the support value (the output membership value) at which membership function i reaches the maximum value of μ_i. It should be noted that the terminology used here is slightly non-standard. In most texts, the term *inference method* is used to mean the combination of the things referred to separately here as *inference* and *composition*. Thus you will see such terms as "MAX-MIN inference" and "SUM-PRODUCT inference" in the literature. They are simply the combination of inference and composition. Occasionally, the reverse terms "MIN-MAX" and "PRODUCT-SUM" are also seen. It seems clearer to describe the two processes separately.

2.5 Fuzzy Applications

The most direct and immediate applications for fuzzy logic have been to the development of control devices. These range from controllers for subway systems and blast furnaces to anti-shake control for video cameras and colour control within copiers. The significant reason for this is that to use traditional control theory one must be able to describe mathematically the system to be controlled.

A famous example is a controller that will balance a pole on a cart whose forward and backward motion can be controlled (see for example, Chapter 8 of Kosko's book [15]). To model the motion requires the solution of a second-order differential equation. If large deviations beyond the vertical are to be described, the approximation of sine theta by theta becomes inaccurate. The more correct equation is nonlinear and needs to be solved numerically. To find a solution to the equation that balances the pole is not really possible, as the solution point is unstable with respect to the system's initial conditions.

On the other hand, it is possible to develop a controller using fuzzy logic without anything more sophisticated than codifying the membership functions for rules such as:

IF pole angle positive medium THEN cart speed positive medium.

This requires experience, but only elementary mathematics; the equations of motion are not needed by the controller. Other complicated control problems can be tackled in this way; for example, M. Sugeno at the Tokyo Institute of Technology is studying the use of fuzzy logic in full size helicopter control systems [12]. This is one of the most difficult control tasks for human operators.

2.6 Fuzzy Logic and Neural Networks

Fuzzy logic complements and parallels neural networks and it is not surprising to see a great deal of research work trying to capitalize on the best characteristics of each. This involves the possibility of cascading the two devices or otherwise merging them. For example, Kosko describes how fuzzy systems behave as associative memories, and it is fairly common to consider using neural networks to learn membership functions of fuzzy sets.

There is also considerable activity trying to understand how fuzzy logic and neural networks are associated with knowledge or reasoning with several conflicting and divergent views. Researchers who specialize in brain function and neurophysics have been emphasizing that the precise models of neurons usually presented in computer science reasoning research have almost no relation to the vastly more complicated real nervous system and that fuzzy rules are much better at describing real physiology.

If we consider neural networks as only numerical processing tools and knowledge based systems as symbolic processing tools then we might regard fuzzy logic as somehow buffering the two, or providing the mechanism to interface the imprecise world with computer precision. From that point of view fuzzy logic seems to be more associated with higher order cognitive processing, while neural networks can learn and are data driven. Fuzziness is easy to understand while neural networks are not; neural networks can learn or adapt while this is more difficult for fuzzy systems.

Fuzzy sets can be perceived as a novel design technology for use in complex and human-oriented applications. Fuzzy logic tries to emulate some properties that humans use in their control and decision processes. In this there arises a need to incorporate many sensors to input information. These will inevitably be needed to gather the vast amount of additional data usually available to humans. Thus it is important not only to talk about fuzzy sets alone, but to become aware of the sensor inputs that are necessary to create algorithms based on fuzzy sets.

Fuzzy logic has had a major impact in Japan. It is interesting to speculate on the reasons for the lack of a general acceptance of fuzzy logic in North America. Membership function subjectivity is probably one important reason. At the elementary level fuzzy logic has a certain apparent imprecision that sometimes puts people off, especially in a university environment. But it is easy to grasp the rudiments of fuzzy logic and it allows the solution of problems in standard ways that would require much more technical mastery to deal with by other means.

2.7 Fuzzy Logic at NASA

NASA has reported encouraging research results in a number of areas using fuzzy logic [13]. One of the more advanced projects is a controller for space shuttle proximity operations, i.e. maneuvering around or keeping position with respect to another object in space. Work has been progressing on a fuzzy-based translational controller which deals with the parameters of azimuth and angle and their respective rates of change, and the range and rate of change of range with respect to another object.

NASA engineers have developed natural language rules to run the controller and are testing them in a multi-vehicle simulation by substituting the fuzzy controller for the simulator's normal human inputs. The rule base was learned from the experience of human operators and the efficiency of the controller was tuned based on flight profiles recorded from actual missions and simulations. One of the main advantages in developing the fuzzy translational controller was that the engineers did not need to construct a detailed mathematical model of the system in advance. Performance was gained through simulation and experience.

The results of simulations have been encouraging, especially in terms of fuel efficiency. In holding position with respect to a target, the fuzzy controller required significantly less acceleration (i.e. smaller increments of position change) than did the human controlled simulation. In overall manoeuvers, the fuzzy controller has shown a 20% to 70% better fuel efficiency than the currently used digital auto pilot and the best simulation runs of human pilots.

NASA is also exploring other applications of fuzzy control in space. Among the projects being considered are the use of inexpensive cameras for constant tracking of objects around the space station. Fuzzy control can contribute to collision avoidance systems, robot arm control and traffic management. At the great distance of interplanetary space where it can take 20 minutes or more to send a signal and receive an answer, robotic systems will have to operate quasi-independently. A fuzzy controller on an unmanned Mars rover vehicle is expected to help the rover avoid obstacles and identify and collect soil samples based on imprecise sensor input and only partially known conditions.

CHAPTER 3

PROBLEM ANALYSIS AND DESIGN

3.1 The Truck Backer-Upper Problem

Backing a truck and trailer to a loading dock is a difficult exercise for all but skilled truck drivers. Normal driving instincts typically lead to erroneous movements and a great deal of practice is required to develop the requisite skills. Because of this, the problem is considered to be severely non-linear and has received considerable attention and interest since the original truck backer-upper paper of Nguyen and Widrow [14].

In Nguyen and Widrow's simulation, the truck only moves backwards with a goal to dock the trailer parallel to the loading dock having the center of the trailer aligned as closely as possible with the center of the dock. They used a two-stage learning process to solve the problem. The first stage involved the training of a neural network emulator of the truck and trailer kinematics. This stage contained 45 hidden Adaline units producing 6 next-state predictions from the current state and steering inputs. The second stage involved the training of a second neural network to control the emulator. This network consisted of 25 hidden units producing 1 steering signal output from the current state inputs. Once the controller learned to control the emulator it was then able to control the actual truck and trailer.

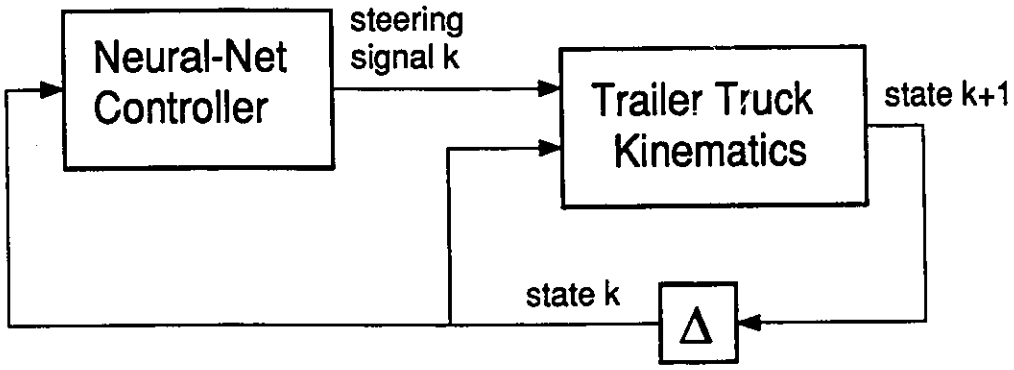


Figure 1 - Overview Diagram

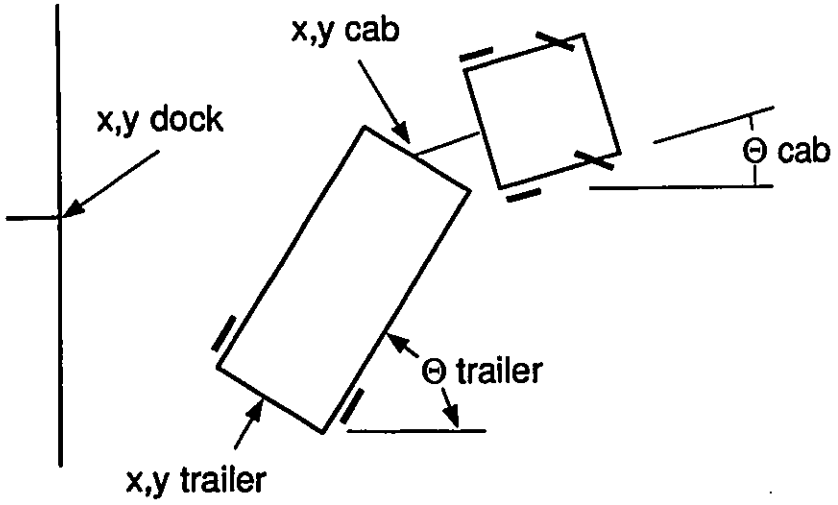


Figure 2 - Truck, Trailer and Dock

Figure 1 is a diagram of the complete system showing how the controller uses the current-state information to produce a correct steering signal for the truck. The state vectors consist of 6 variables describing, in Cartesian coordinates, the position and orientation of the truck and trailer relative to the loading dock (see **Figure 2**).

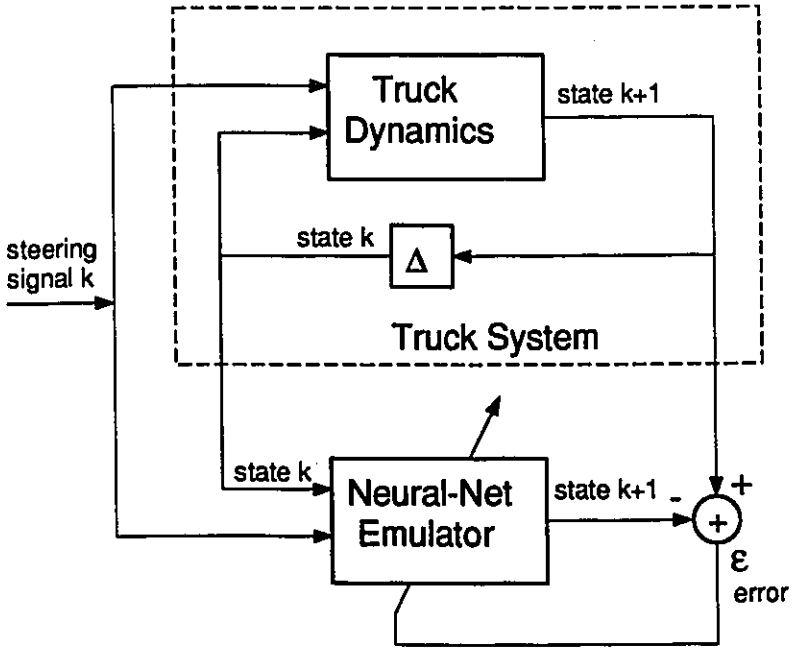


Figure 3 - Training the Emulator

Figure 3 shows a block diagram of the process used to train the emulator. The truck backed up using randomly selected steering signals. After training on a wide variety of state vectors and steering signals, the emulator *learned* the kinematic behaviour of the truck and trailer system and was then used to train the controller.

This training process was quite laborious, requiring about 20,000 backups in all to train the controller; however, it performed well in backing up the truck from difficult initial starting configurations. The scheme employed is not really unsupervised learning since the sequence of learning trials were carefully designed and controlled; however, the proper steering signals were never provided but were discovered through time.

Kosko [15] attacked the problem with both neural networks and fuzzy logic. Since he had difficulty training the emulator network, it was replaced by simple kinematic equations. His emulator is based on a cartesian coordinate analysis similar to that of Nguyen and Widrow and is shown in **Figure 4**. The four state variables x, y, ϕ_t and ϕ_c determine the position of the truck and trailer system uniquely in the plane. Referring to **Figure 4**, x and y are the Cartesian coordinates of the rear end of the trailer while u and v are the coordinates of the joint. ϕ_t is the angle of the trailer with the horizontal and ϕ_c is the relative angle of the cab with the trailer. θ is the steering angle and β, the fuzzy output variable, is the angle of the trailer updated at each step.

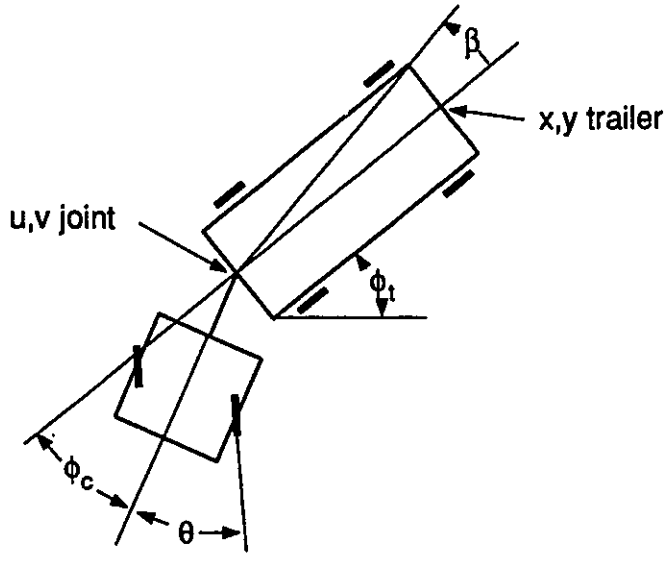


Figure 4 - Truck and Trailer System

With the output value β computed by the fuzzy inference engine, the trailer position (x,y) moves to a new position (x',y'):

$$x' = x + d\cos(\phi_t + \beta)$$

$$y' = y + d\sin(\phi_t + \beta)$$

where d is a fixed backing distance.

The joint of the cab and trailer (u,v) also moves to a new position (u',v'):

$$u' = x' - l\cos(\phi_t + \beta)$$

$$v' = y' - l\sin(\phi_t + \beta)$$

where l is the trailer length. The directional vector (dir$U$, dir$V$) which defines the cab angle ϕ_c is updated to (dirU', dirV'):

$$\text{dir}U' = \text{dir}U + \Delta u$$

$$\text{dir}V' = \text{dir}V + \Delta v$$

where $\Delta u = u' - u$ and $\Delta v = v' - v$. The new directional vector (dirU', dirV') defines the new cab angle ϕ'_c. Then the steering angle is determined as $\theta = \phi'_{c,h} - \phi_{c,h}$, where $\phi_{c,h}$ is the cab angle with the horizontal. The fuzzy rule base contains 105 "hand-crafted" fuzzy rules out of a possible 735 rules. Kosko reports successful control of the truck and trailer even in jack-knifed positions.

Plumer [16] goes one step further and examines a computationally efficient method of modifying the truck backing system to permit vehicle control in the presence of obstacles using potential fields. He designs a potential field which peaks at the surface of obstacles and has a minimum at the proper vehicle destination (docked). A feed-forward neural network is then used to control the steering of the vehicle using local field information. The technique is interesting since the truck is only trained in an obstacle free space but is then capable of docking in the presence of obstacles which may be stationary or movable.

Koza [17] applied a genetic programming technique to evolve a computer program to control the truck. This technique is especially appealing since neither the size nor shape of the solution were pre-specified. Each of 1000 individual programs from each generation were simply scored on how well they backed up the truck from 8 different initial configurations. By generation 26, typically, a control strategy capable of successfully docking the truck from each of the test cases had evolved.

3.2 Problem Analysis using Polar Coordinates

The studies mentioned above were, of course, all simulations employing a computer model of the truck dynamical system. In addition, the latter studies all followed Nguyen and Widrow's lead in analyzing the problem using cartesian coordinates. This is unfortunate, since the analysis is significantly simpler with polar coordinates.

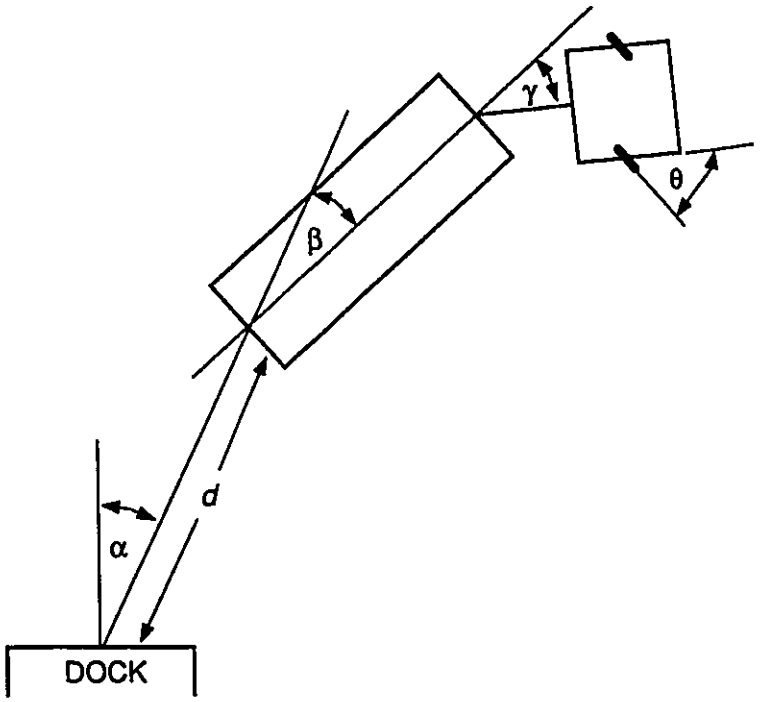


Figure 5 - Analysis using Polar Coordinates

Referring to **Figure 5**, the position of the trailer with respect to the dock can be completely described on a two dimensional surface using three state variables. α is the angle between a vertical line from the dock point and the line from the dock point to the rear of the trailer. β is the angle between the line from the dock point to the rear of the trailer and the center-line through the trailer. d is the distance from the dock point to the rear of the trailer.

The control variable is γ, the angle between the cab and the trailer center-lines. Additionally, it can be seen that the steering wheel angle, θ will completely determine γ. All angles in **Figure 5** are shown in their positive (clockwise) orientation.

The problem of backing up the trailer to the dock (**Figure 6**) can be stated mathematically as reducing the values of α, β and d so that:

$$\alpha = \beta = d = 0$$

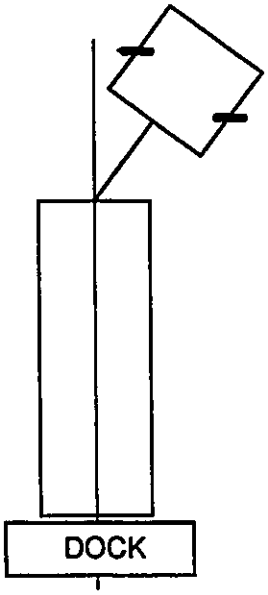


Figure 6 - The Fuzzy Truck successfully docked

There are obviously many ways of reducing these variables to zero. Assuming that we are reversing towards the dock, or at least diminishing α and β, we can essentially ignore d and assume that it will eventually go to zero. How we diminish α and β will affect the trajectory the trailer follows in backing up to the dock. Many different trajectories will accomplish the task.

If α goes to zero more quickly than β, then the trailer will follow a highly curved trajectory whereas if β goes to zero more quickly than α, the trailer will approach the dock more directly (see **Figure 7**). A compromise control strategy and one that is easy to implement is to control γ to maintain $\alpha = \beta$. In this fashion, both α and β approach zero at the same rate so that the trailer follows more or less a circular trajectory in approaching the dock.

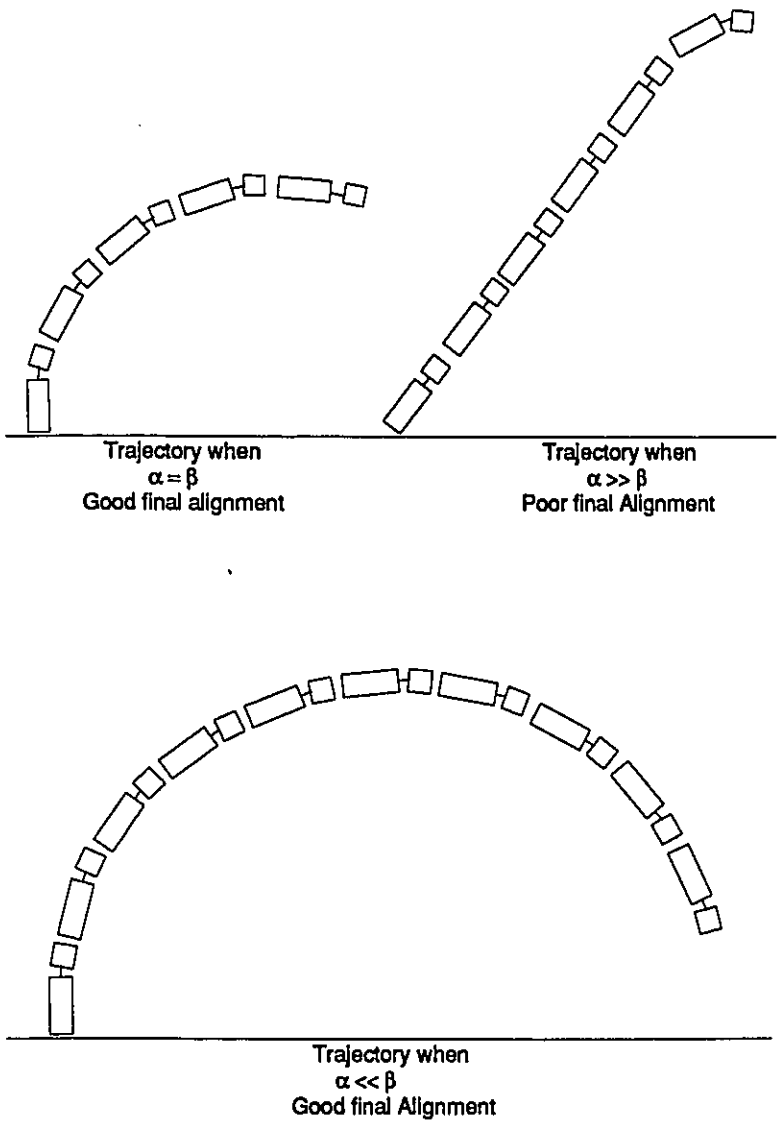


Figure 7 - Docking Strategies

In the spirit of fuzzy control design, no further mathematical analysis is required. It is important to note the simplicity of the analysis accomodated by the adoption of the polar coordinate system. The beauty of fuzzy design is that only a *qualitative* analysis of the problem need be carried out. The remainder of the analysis for this problem is somewhat subjective but fortunately rather intuitive. Referring to **Figure 5** again, if α is larger than β (we want to increase β) then we move the cab to the *left* (decrease γ) by steering right to correct. Notice that this strategy is also correct in the other quadrant (left of the vertical from the dock).

Forward motion is used only when the trailer is in a position from which it would be awkward or difficult to approach the dock with proper alignment. In moving forward, we again ignore d and assume that the truck and trailer will eventually straighten out sufficiently to allow the backing up strategy to succeed. In addition, we can exploit the symmetry of the system by using the above intuitive rules in reverse; i.e., we will still maintain $\alpha = \beta$, but now move the cab to the *right* to increase β by steering right. Notice that to increase β going forward or backward we steer right.

In reality, the forward motion of the truck and trailer is limited by the maximum length of the tether and the cab is additionally limited by a *jack-knife* angle to the left and right so we need to design rules to take corrective action on reaching these extreme positions. Otherwise, the cab angle is essentially *irrelevant* to the docking strategy - the steering signal determines the desired change to the trailer angles. This is the only relationship which makes the task difficult since the problem is essentially linear in polar coordinates. If the cab is jack-knifed to the right, and we need to rotate the trailer in a clockwise direction (increase β), then veering to the right (the correct action) will not result in clockwise motion until the cab has manoeuvred itself around to the other side of the trailer. This is where inexperienced drivers typically get into trouble - they expect immediate response to what appears to be the correct control action. This is basically what makes the truck backer-upper problem highly non-linear.

3.3 The Truck Systems

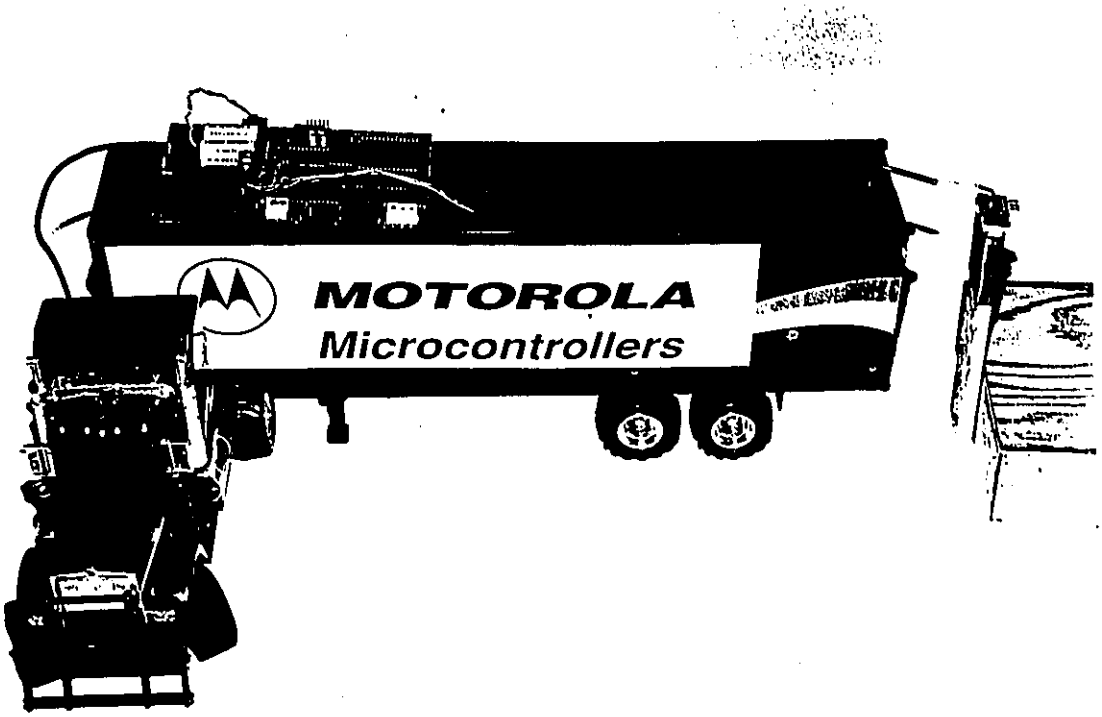


Photo 1 - The Fuzzy Truck
The MCU board can be seen on top of the trailer. The thin wires attached to the dock enter the truck through a rotating bent tube and are taken up on a spool inside the trailer (see Photo 4).

A toy electric truck and trailer was selected as a testbed for the fuzzy controller. The particular model selected measures about 28" long and 7" high (see **Photo 1**). It contains two small electric motors, one to control steering and a slightly larger motor connected through a gear train to the cab drive wheels.

The original controller box supplied with the truck and trailer was discarded and the cable connected to the MCU controller board on top of the trailer as shown in **Photo 2**. The original steering motor was replaced with a proportional servo, of the type used in radio control models, in order to provide more precise steering control but the main drive motor was used as supplied. Some plastic was removed from around the steering mechanism in order to allow extended steering wheel movement and the original front wheels were later replaced by racing wheels from an R/C model shop.

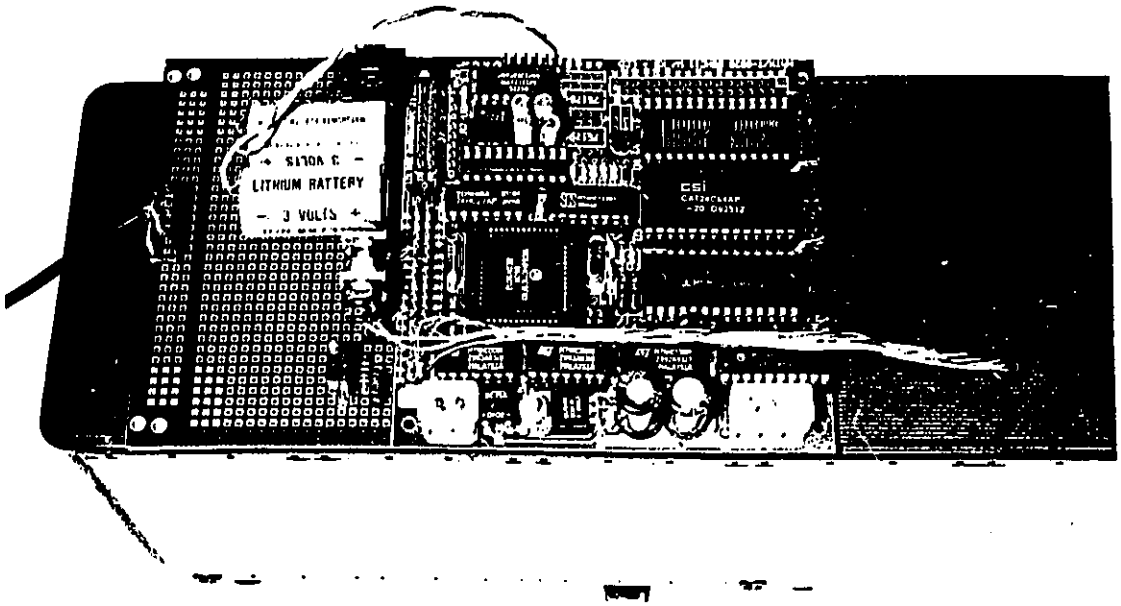


Photo 2 - The NMIT MCU Board
Note the cable at the top of the board going inside the cab. The wires leading inside the trailer are connected to the position sensing potentiometers.

This modification was found necessary after initial steering tests since the original truck had very limited steering capability. A small 6 volt sealed lead-acid battery was fitted into the cab and a jack added for recharging (see **Photo 3** - inside the cab). This particular truck has the added advantage of being assembled with small screws, facilitating modifications.

The inside of the trailer has ample room for the position sensing components and the controller board, although for demonstration purposes the board has been left on the top of the trailer. As previously mentioned, the current position sensing system measures the three state variables α, β and d and the control variable γ directly using potentiometers to measure the angles and a thin wire to measure the distance d. Actually, two wires are used in order to feed the signal from the potentiometer on the dock to the controller in the truck. The wire is wound onto a spool inside the trailer that is tensioned by an elastic. The rotation of the spool is measured by a fourth multi-turn potentiometer (see **Photo 4** - inside the trailer).

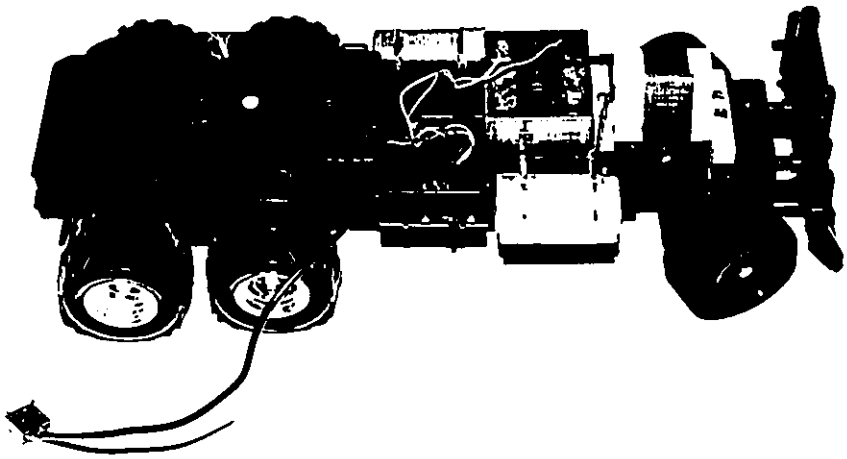


Photo 3 - Inside the Cab
Note the Heathkit servo motor which controls the position of the front wheels. Also visible is the 6 volt Dryfit battery and the charging jack at the bottom of the cab.

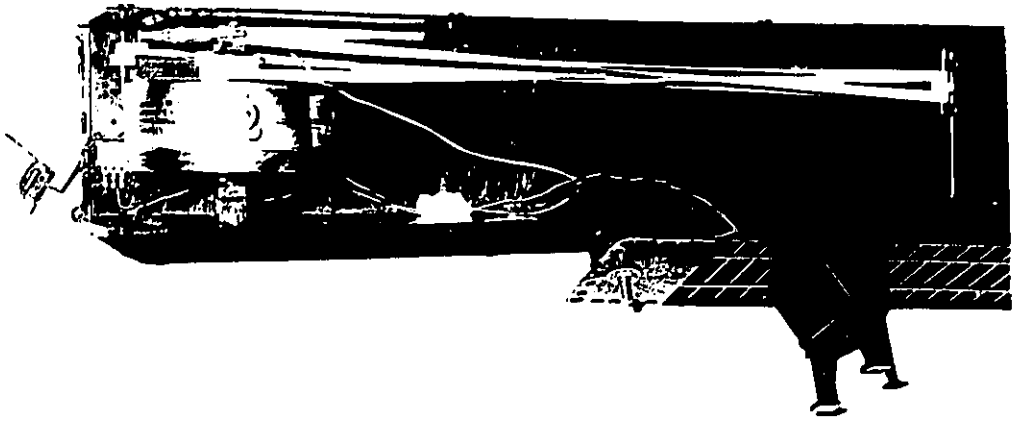


Photo 4 - Inside the Trailer
Note the large take-up spool and potentiometer on one side. The elastic cord makes four passes down the length of the trailer. Also visible is another potentiometer on the ceiling of the trailer used to measure β.

The mechanical components used in the position sensing system can only be described as "Rube Goldberg". The parts were all obtained from the author's "junk box" and a partial list of materials reads as follows:

1. one very old mechanical mouse (supplied potentiometers, roller-bearing for brass tube and bearings used for fabric elastic)
2. one fruit drops tin (used as wire take-up spool)
3. one Bournes multi-turn potentiometer (fitted to the tin with a rubber grommet)
4. one linear-taper potentiometer mounted through the floor of the tr..iler (replacing the original plastic hitch)
5. one-half of an empty thread spool (used as a take-up flange for the fabric elastic)
6. one old brass carburettor jet (hollow bearing to allow wire to pass from inside of take-up spool)
7. a two to three foot piece of ¼" fabric elastic
8. a 2½" length of brass tube from a ball-point pen insert (tube is bent at a 90° angle and inserted into the roof-mounted ball bearing)
9. assorted Mechano flanges, nuts and bolts, etc.

The state and control variables are measured as follows. Angle α is measured by one of the small mouse potentiometers mounted at the dock. This potentiometer and the one used to measure β, being designed for use in a mouse, move very freely. A small finish nail bent at a 90° angle was forced, by heating, through the center of the plastic potentiometer. Two pieces of enamel wire approximately 40 AWG and about three feet long were soldered to the potentiometer and taped to the finish nail. In this fashion, as the tensioned wires rotate about the dock, the potentiometer follows and its position is fed to the on-board controller via the wires.

The two wires pass through the bent brass tube and into the truck where they are taken up on the rotating spool. Inside the trailer, a small plastic gear forced onto the tube turns another small mouse potentiometer. In this fashion, angle β is measured since the tube will also follow the movement of the tensioned wires.

A multi-turn potentiometer on one side of the spool measures d, the distance from the back of the trailer to the dock. To the other side of the spool is fastened a flange to which is fastened one end of the fabric elastic. This elastic makes four passes down the length of the trailer before being fastened to the spool support. Two sleeve bearings (from the mouse again) were used at either end to allow smooth stretching of the elastic over its length. This elaborate tensioning system was found necessary in order to produce near-constant tension over the three foot range of the wire cable.

A final linear potentiometer was mounted through the floor of the trailer and serves as the trailer hitch. The screwdriver slot in this potentiometer mates with a thin metal bracket on the cab. As the cab rotates under the trailer, this potentiometer follows and provides a measurement of the control variable γ. Wires attached to the four potentiometers pass through a grommet on the roof of the trailer to the MCU controller board.

3.4 An Optical Sensing System

Because of the obvious limitations of this mechanical system, an optical based sensing system was also designed and tested. The optical interface is based on a Symbol Technologies SE1000 miniature bar code engine measuring approximately 40mm by 25mm by 20mm which is mounted on a small servo motor controlled by the MCU (see **Photo 5**). This bar code engine employs a laser diode emitting a visible red beam ($\approx$ 680 nm) and vibrating mirror which scans a 45° sector at a rate of 36 Hz. The reflected beam is sensed by a photodiode, amplified and thresholded. This digital signal can readily be processed using the input capture feature of the MC68HC11 MCU.

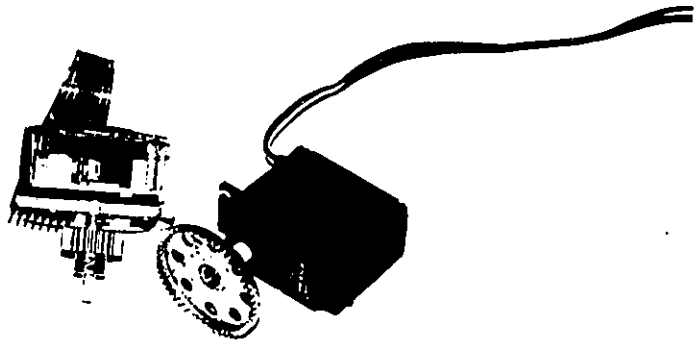


Photo 5 - The Laser Bar Code Engine
The vibrating mirror is just below and behind the photodiode array on top of the unit. The laser diode is in the housing to the lower left. The Mechano gear under the unit meshes with a servo inside the trailer.

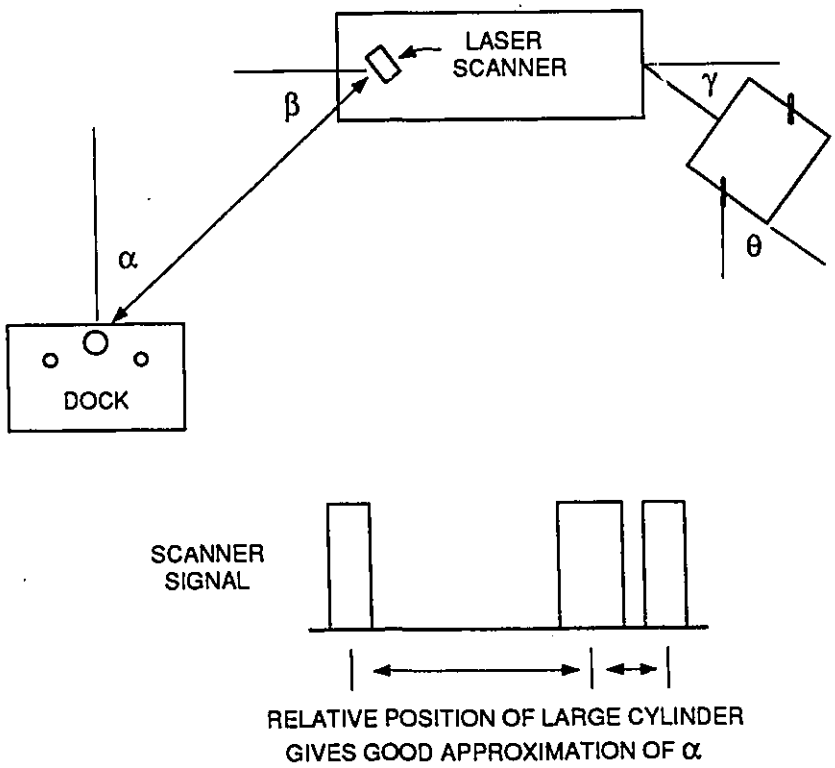


Figure 8 - The Optical Sensing System

Although the SE1000 has been designed primarily for short range reading of bar codes (< 1 m), its range can be extended to 3 metres or more using retro-reflective materials at the docking position. The reflective pattern is two-dimensional in design employing a center post in front of the pattern to allow determination of angle α (see **Figure 8**).

Angle β can be determined directly from the orientation of the servo motor. This servo motor will continuously track the pattern on the dock as the truck moves using a separate interrupt routine. Distance d from the back of the trailer to the dock can also be approximated by measuring the width of the large cylinder as seen by the scanner. For the most part, the truck operates at constant speed, slowing down as it approaches the dock closely. Thus the most critical measurement of d occurs at very close range, where the resolution of the scanner is at its maximum.

3.5 The Microcontroller Unit

The Motorola MC68HC11 is an 8-bit microcontroller unit (MCU) fabricated in high-density complementary metal-oxide (HCMOS). It contains very powerful on-chip peripheral capabilities which simplify hardware design [18]. The MC68HC11E9 version used in this project includes the following on-chip resources:

1. 8K ROM (monitor/debugger)
2. 512 bytes RAM
3. 512 bytes EEPROM
4. 8 channel A/D converter
5. asynchronous serial communications interface (SCI)
6. synchronous serial peripheral interface (SPI)
7. 16-bit free running timer with 4 input capture and 5 output compare lines
8. 8 bit pulse accumulator system
9. 5 8-bit peripheral I/O ports

Not all of the I/O port lines are available depending on the mode of operation and the use of other resources such as A/D and serial I/O. The MCU is operated in the expanded chip mode for use in the fuzzy truck in order to add external memory for the fuzzy logic rule base. In this mode, two of the peripheral ports (port B and C) are used to expand the address and data bus externally.

In addition, three of the port A lines are used as output compare lines for controlling the steering servo and the H-bridge motor drive, two of the port D lines are used for the serial interface to the PC (SCI) and four of the port E lines are used as A/D inputs for the position sensing potentiometers. This leaves several other I/O lines for future expansion.

The original board used for the controller was the Motorola MC68HC11 Evaluation Board Unit (EVBU). This board provides serial capability and includes an external clock/calendar chip. A prototype area on the board provides space for custom hardware. Although this area could have been used to expand the address and data buses externally, a similar board available from New Micros Inc., the NMIT-0020, was later adopted. This board is slightly larger and is supplied with external bus capability including three 28-pin sockets for memory chips. The MC68HC11 chip provided with this board contains a 12K byte FORTH interpreter which will be used in future expansion of the project. The original fuzzy logic engine and interface software written in assembler was used by replacing this chip with the E9 version from the original Motorola board.

All of the position sensing potentiometers used are linear taper and all three terminals are directly connected to the port E A/D lines, simplifying the normalization process. The only exception is the potentiometer located on the dock. For practical reasons only two wires were used in the tether necessitating the use of a pullup resistor to Vcc. This resistor is actually a small potentiometer which is adjusted to provide near linear operation of the potentiometer on the dock.

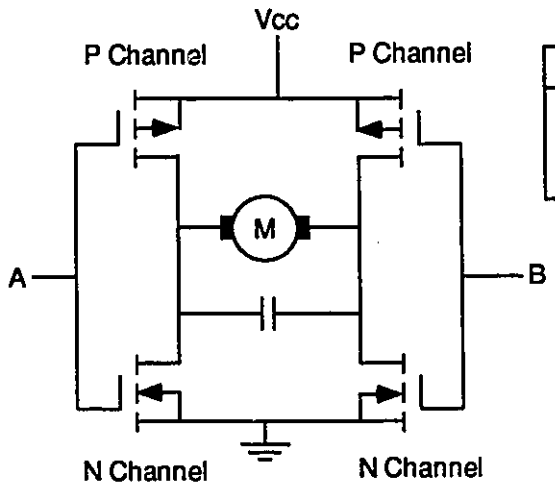
3.6 Drive Motor Control

Several design options were considered for the H-bridge motor controller. Since motor power is supplied by on-board batteries, it is essential to implement a power conservative design. For efficiency and simplicity of design it is hard to beat MOSFET devices. These devices, being voltage-controlled, can be switched on and off directly from the MCU's CMOS output lines. In addition, modern devices can be found with very low on resistance which means that very little power is dissipated as heat by the device.

There are essentially two design alternatives for an H-bridge motor controller. Because of the superiority of n-channel over p-channel CMOS devices, earlier designs used n-channel devices in all legs of the bridge. The only problem with this design is that when an n-channel device is used for the high-side switch, the turn-on gate voltage must be pulled higher than the positive rail. Either we must use a separate battery or add additional circuitry to the gate drive network in the form of a charge pump. Although this complicates the design considerably, many manufacturers solve this problem by integrating the required subsystems on a single motor-driver chip. For instance, Motorola's MPC1710A motor driver chip requires only three small external capacitors for the charge pump. Operating from a 7 volt supply, it can deliver up to 1A of current with a 0.4 ohm on-resistance when sourcing current and 0.2 ohm resistance when sinking current. Another popular motor driver H-bridge using bipolar transistors is the L293D manufactured by SGS Thompson.

Advances in the design of MOSFET devices has led to the availability of very low on-resistance devices, both n-channel and p-channel. The design of an H-bridge using p-channel devices as the high side switches and n-channel as the low-side switches is trivial (see **Figure 9**). Referring to the schematic, it can be seen that no components other than the four MOSFET devices are required. The gates are driven directly from the output-compare lines of the MCU. The devices used are International Rectifier IRFZ30 (n-channel) and IRF9Z30 (p-channel). These are 50 volt high current devices (30A and 18A respectively) with rated on resistances of 0.05 ohms and 0.14 ohms respectively.

The low-threshold capabilities of these devices permit their operation from the 6 volt battery pack used in the fuzzy truck. In-circuit measurements taken during operation indicated an actual on-resistance of 0.17 ohms for the n-channel devices and 0.33 ohms for the p-channel devices.



	A	B
FWD	1	0
REV	0	1
OFF	0	0
OFF	1	1

Figure 9 - The H-Bridge Motor Controlle

3.7 Steering Control

As described earlier, the original steering system was discarded in favour of a model servo motor. Like most servos used in model aircraft, boats or other vehicles, its position is controlled proportionally by a pulse-width modulated (PWM) signal. Referring to **Figure 10**, the servo arm can be rotated through approximately 200 degrees by varying the width of the control signal. This simplifies the control software considerably, since we need only indicate the steering wheel position and the servo will faithfully respond. The software is further simplified by using the output-compare feature of the MC68HC11 MCU.

The main timer in the MC68HC11 counts continuously from $0000 (out of reset) to $FFFF. It then rolls over to $0000, sets an overflow flag and continues to count up. A programmable prescaler allows the user to select one of four clocking rates to drive this main timer. The default fastest clocking rate, used in this project, causes the main timer to count at the E-clock rate of 2 Mhz. This results in a timer resolution of 500 ns and a timer range of 32.77 ms between overflows. This is appropriate since both the steering servo motor and drive motor are controlled within a 14ms cycle.

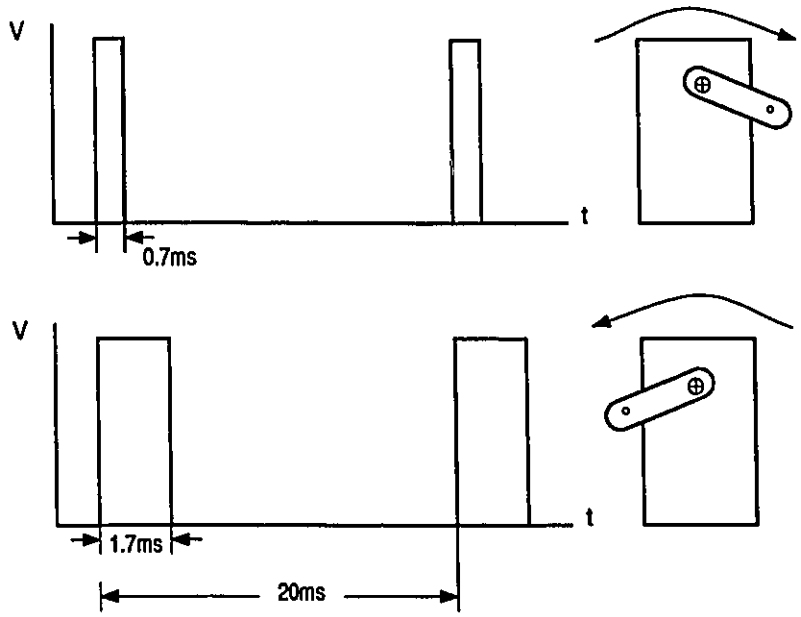


Figure 10 - Controlling a Servo with PWM

3.8 Output-Compare Feature

The operation of the output-compare feature is best explained by examining a code fragment from the fuzzy truck assembler code (see **Figure 11**). The initial assembler code at the top of the figure is executed once only to initialize the 16-bit main timer and output-compare lines. This code establishes the jump vector for the interrupt routine that controls both the steering motor and drive motor. Only Output Compare 1 (OC1) will generate an interrupt. Its period of operation is established as 14ms ($6FFF E-clock cycles). OC1 is initialized so that it will turn on the other three Output Compare lines (OC2, OC3 and OC4). OC2 controls the pulse width for the steering servo, OC3 controls the drive motor pulse width in the forward direction and OC4 controls the drive motor pulse width in the reverse direction. In actual fact, OC3 directly drives one gate pair of the H-bridge while OC4 drives the other gate pair. In order for the motor to be energized, one of these lines must be pulled low.

```
* setup OC1 and OC2
*
        LDAA    #$7E
        STAA    OC1VEC
        LDD     #OC1SVC
        STD     OC1VEC+1        setup OC1 vector
        LDD     #$7080
        STAA    OC1M,X
        STAA    OC1D,X          OC1 will turn OC2/3/4 on
        STAB    TCTL1,X         setup OC2 to turn off
        LDD     TCNT,X
        STD     TOC1,X          wait a full cycle
        BSET    TMSK1,X $80     enable OC1 int
*
*
* OC1 interrupt service routine generates pulse for
* steering servo and main engine
*
OC1SVC  LDX     #REGBAS         load reg base
        BCLR    TFLG1,X         $7F clear int flag
        LDD     TOC1,X          get last compare
        ADDD    STEER
        STD     TOC2,X          update steer pulse width
        SUBD    STEER
        ADDD    #OC1_PER
        STD     TOC1,X          update OC1 cycle
        SUBD    SPEED
        TST     DIRN
        BPL     FWD
REVSTD  TOC4,X  update engine pulse
        LDAA    #$86
        STAA    TCTL1,X         setup OC4
        RTI
FWD     STD     TOC3,X          update engine pulse
        LDAA    #$A0
        STAA    TCTL1,X         setup OC3
        RTI
```

Figure 11 - Output Compare Assembler Code

A single control cycle operates as follows (refer to the interrupt service routine code fragment OC1SVC in **Figure 11**):

1. entry to the interrupt routine is generated by a main timer compare on OC1. This compare has also automatically caused a positive transition on OC2, OC3 and OC4. Note that this initiates a steering pulse but *not* a drive motor pulse. The first step is to clear the interrupt flag and retrieve the current value of the main timer (the compare value previously stored).

2. the current defuzzified and renormalized steering pulse value (in STEER) is added to the current timer value and stored in the OC2 timer compare register. When the main timer reaches this value, OC2 will be turned off automatically, thus generating a pulse of appropriate width to the steering servo.

3. next the value of OC1's period (14ms) is added to the current value of the main timer and this updated value stored in the OC1 timer compare register. This will eventually cause the cycle to repeat by generating another OC1 interrupt.

4. we now *subtract* the defuzzified and drive motor value in SPEED from the current timer value. Depending on whether we are in Forward or Reverse mode we establish OC3 or OC4 to turn *off* when the main timer reaches this value. This will then cause the drive motor to turn *on* in the appropriate direction and remain on for the remainder of the 14ms period.

In this fashion the steering servo and drive motor are each controlled by individual PWM processes managed within a single interrupt cycle, and service routine. The entire process is simplified by the output capture feature of the MCU.

CHAPTER 4

FUZZY DESIGN

4.1 Fuzzy System Design

The input variables for this problem have been previously described as follows:

- α the angle between a vertical line perpendicular to the docking point and the line from the dock point to the rear of the trailer.
- β the angle between the line from the dock point to the rear of the trailer and the centre-line through the trailer.
- d the distance from the dock point to the rear of the trailer.

The control variable is γ, the angle between the cab and the trailer centre-lines. As indicated previously, the steering wheel angle θ completely determines γ and is used as the main control variable.. The usual approach in fuzzy design would lead to a set of 343 rules for this problem (7α X 7β X 7γ). To keep the rule base smaller we exploit the symmetry of the problem by defining a new variable *AB* as the difference between α and β. The current fuzzy system defines 3 input fuzzy variables called *AB* (α - β), *GAMMA* (γ) and *DIST* (*d*) and 3 output variables called *STEER* (θ), *SPEED* and *DIRN*. Both *AB* and *GAMMA* have 7 values each, requiring 49 rules (for all combinations) for the steering function. The truck speed is determined only by its distance from the dock and is controlled by three additional simple rules (see Figure 20).

Early in 1992, Motorola announced the availability of a *Fuzzy Logic Educational Kit*. This kit contains a comprehensive Fuzzy tutorial and Motorola software (distributed as *freeware*) which implements a rudimentary PC based *Knowledge Base Generator* (**KBG**) and *Fuzzy Engine* for the MC68HC11 MCU. The engine requires approximately 256 bytes of memory leaving the remaining 256 bytes of on-board EEPROM for user code.

The knowledge base generator and fuzzy engine designed by Motorola allows a maximum of 8 input variables and 4 output variables. The input membership functions are implemented as simple trapezoidal shapes while the output functions are singletons. This implementation simplifies processing while conserving space.

4.2 The Knowledge Base Generator

The program KBG.EXE supplied by Motorola creates and edits a knowledge base, runs a software simulation of an inference engine and uses the simulation to create a 2-D plot of the control surface described by the knowledge base. The program acts as a pre-processor to generate a knowledge base for the MC68HC11 inference engine using natural language inputs. Output from the program is an assembly code file containing the fuzzy knowledge base in the form of Form Constant Byte (FCB) assembler directives scaled from 0 to 255 (see **Appendix** B).

The current version of KBG.EXE is compiled to accept:

1. 8 inputs and 4 outputs
2. 8 membership functions per input/output
3. 1024 rules
4. any number of inferences per rule
5. 4 points per input member (trapezoid)
6. 1 point per output member (singleton)

The program is menu-driven and defines name fields for both the input and output variables as well as the units of measurement. The units of measurement contain minimum and maximum values and scaling is done automatically by the program. This considerably simplifies the definition of the membership functions and rules which can be described using familiar units of measurement such as degrees for angles and inches for distances.

Having defined the input and output variables, the next step is to define the membership functions. In the case of the fuzzy truck, the angular variables α, β and γ were defined to have seven membership functions each, namely: *Small*, *Medium* and *Large* displacements on either side of center (*Left* and *Right*) and *Zero* for center displacement. For smooth operation, these functions were defined as overlapping trapezoids. The trapezoids are defined using four x-coordinate points (the program "connects-the-dots" to produce the trapezoids). In special cases such as squares and triangles, some of the points are identical, but in all cases four points are defined for each membership function. Note that, as previously mentioned, the units of measurement defined for each variable are used when inputting trapezoid function points. The program displays both scaled and normalized function values as they are input and allows the user to edit the values. Each of the membership functions also has a name such as *Negative_Large* or *Positive_Small* which simplifies input and edit of rules. The definition of output membership functions is very simple since outputs are represented as singletons (vertical lines).

Rules are now entered using a simple English language structure. In fact, since there may be many rules, the program simplifies input using menu choices. After typing "IF" as the beginning of the first rule, the program displays a menu list of all input variables which the user may select using menu choices. Upon selection of an input variable, the program then adds "IS" to the rule string and displays a menu list of the defined membership functions. After selection, the program then asks for a choice of "AND" or "THEN".

If "AND" is selected, another input variable is processed. This repeats until "THEN" is selected in which case a menu list of output variables is presented. Again, upon selecting an output, a menu list of the membership functions for this variable is presented and a choice of "AND" or "END" follows. Several output inferences may be entered. The rule is complete when "END" is selected. In this way, many rules can be entered quickly using single keystrokes. Each rule as implemented in the Fuzzy Truck requires only 2, 3 or 4 bytes of memory. The sign bit (MSB) of each byte is used to indicate the switch from consequent (IF) to antecedent (THEN) and the beginning of a new rule. The remaining bits indicate both the variable and its membership function.

Appendix B shows the assembly language listing of the knowledge base produced by KBG.EXE. Note that the variable names and membership function names appear as comments in the listing. For ease of processing, the inference engine assumes 8 membership functions per variable, unused entries are zero. Note the four bytes for each input function defining a trapezoid and the one byte required for output singletons. The rules are harder to follow since no comments appear in the listing. An MSB of 1 (8 or larger in the first hexadecimal digit) indicates the last byte of a precedent or antecedent. The final byte of $FF is used to indicate the end of the rule base. Also note the number of inputs and outputs at the end of the listing. These parameters are used by the inference engine. Shown in **Appendix A** is the rule base listing produced by the knowledge base generator program KBG.EXE.

4.3 The Inference Engine

Although the user can work with scaled values in building the knowledge base with KBG.EXE, in programming the main assembly language routine we must obviously provide code to normalize the inputs since the fuzzy inference engine assumes that all inputs and outputs are normalized 8 bit values (0 - $FF). The user code for the Fuzzy Truck shown in the assembly language listing in **Appendix C** performs this scaling function in the main loop before calling the inference engine. Note that the MC68HC11 A/D system continuously performs successive-approximation A/D conversion using a charge redistribution technique; thus the user may simply read the value of a register at any time to obtain the current value of a variable. The code fragment shown in **Figure 12** loads the value of β (*BETA_AD*), subtracts the value of α (*ALPH_AD*) to produce the fuzzy variable *AB_FUZ*. In this case the normalization is complex since both of these variables may be in the range of 0 to $FF. The new variable is scaled by adding $7F and checks are performed for underflow, in which case the new variable is set to zero, and overflow, in which case the variable is set to the allowed maximum of $FF. Note that both γ (*GAMM_AD*) and *d* (*DIST_AD*) are already in normalized form and no conversion code is required.

```
* Normalize I/p parameters
*
IP_NORM  CLRA
         LDAB    BETA_AD,X
         STAB    BETA+1
         LDAB    ALPH_AD,X
         SUBD    BETA          subtract beta from alpha
SCALE    ADDD    #$7F          add 127
         BPL     BIG_AB
         CLRB                  underflow
         BRA     SAVE_AB
BIG_AB   TSTA
         BEQ     SAVE_AB
         LDAB    #$FF          overflow
SAVE_AB  STAB    AB_FUZ
         LDAA    GAMM_AD,X
         STAA    GAM_FUZ
         LDAA    DIST_AD,X
         STAA    DIST_FUZ
```

Figure 12 - Normalization of Input Variables

At this point the fuzzy inference engine is called. Each fuzzy input is processed in turn. The fuzzy input value is projected onto the associated membership functions to develop a grade of membership for each function. Having *fuzzified* all inputs, each of the rules is then evaluated in turn. The grades of antecedent variables are determined by selecting the *smaller* value of the grades of the inputs (*determine the MIN).* The grades of the consequent variables are only updated if the grade of membership associated with the antecedent part is *higher* than the output variables current value (*determine the MAX).*

4.4 Defuzzification

The final step for the fuzzy inference engine is the only computationally complex calculation, the *fuzzy centroid* defuzzification process. Although the simpler *maximum-membership* defuzzification scheme could have been used to considerably speed up this process, as indicated earlier this scheme can lead to problems due to the possible generation of multiple modes. Again, the processing power of the MC68HC11's instruction set performs the centroid defuzzification in well under 50μs. For each fuzzy output the program loops over its membership functions to calculate the sum of the fuzzy outputs and the sum of the weighted fuzzy outputs (the product of the fuzzy output and the output singleton position). An unsigned 8-bit multiply instruction (MUL) and the add with carry instruction (ADC) help to make this process efficient. The final divide of these two sums produces the defuzzified output. An integer divide instruction (IDIV) is used to determine the result if the numerator (the sum of weighted fuzzy outputs) is not larger than 16 bits while a fractional divide (FDIV) instruction following a test for rounding is used for a maximum 19 bit numerator. In either case both divide instructions execute in 41 machine cycles.

Upon return from the fuzzy inference engine, the defuzzified outputs are scaled by the code fragment shown in **Figure 13** and stored for use by the interrupt service routine described in the previous section. The timer values used for the steering servo were found by experimentation to have a maximum of $1000 and a minimum of $0900 while the speed timer values have a maximum of $7E00 and a minimum of $5F00.

The code fragment shown accomplishes scaling of the defuzzified steering value using logical shift left instructions (LSLD) to multiply the defuzzified value by 8. The defuzzified speed and direction values do not need to be adjusted as they are already in normalized form. Note that the direction variable (*DIRN*) is only updated if a change has occurred, corresponding to either the outer or inner ring of the rule matrix (see **Figure 17**).

```
* Adjust fuzzy outputs
*
* Steering
*
ADJ_OP   CLRA
         LDAB    STR_FUZ
         LSLD
         LSLD                    range of $07XX
         LSLD
         ADDA    #$0A            max of $10XX
         STD     STEER
*
* Speed
*
         CLRA
         LDAB    SPD_FUZ
         LSLD
         LSLD
         LSLD                    range of $1FXX
         LSLD
         LSLD
         ADDA    #$5F            max of $7EXX
         STD     SPEED
*
* Direction
*
         LDAA    DIR_FUZ
         BEQ     IP_NORM         no change if zero
         STAA    DIRN
         BRA     IP_NORM         loop to begin
```

Figure 13 - Scaling of De-Fuzzified Output Variables

4.5 Experimental Design

Having built and tested the mechanical and electronic components of the truck, the first step is to calibrate and normalize the analog inputs from the four position sensing potentiometers. In this case it is much easier and more flexible to accomplish the normalization within the fuzzy subset membership functions. Since the fuzzy engine used in the design allows a maximum of 8 membership functions per variable, it seems appropriate to have 3 functions at varying angles on either side of a central zero point. As a side note, an additional design using only 5 membership functions (2 on either side of zero) was also tested but produced rougher control that occasionally led to docking problems. The increase to 7 membership functions adds an insignificant amount of processing time to the total cycle time of the fuzzy inference engine and produces considerably smoother control action.

ANGLE	-90	-60	-45	-30	0	30	45	60	90
α	46	5E	68	71	80	8C	91	96	9E
β	30	4B	58	65	80	9B	A8	B5	D0
γ	30	4B	58	65	80	9B	A8	B5	D0

Figure 14 - Calibration Values of A/D Registers

Each of the potentiometers was calibrated by rotating through 4 angles on either side of zero and taking readings of the value returned in the A/D registers of the MCU (see **Figure 14**). In the case of γ and d, these calibration values were then used to establish the initial membership function mid-points. Note that the minimum and maximum values defined for γ represent the jack-knifed position of the cab (90 degrees on either side of zero). Not shown are the distance values which simply range from 0 to 36 inches over the normalized range ($00 to $FF).

This scaling feature of the knowledge base generator simplifies the user code required and makes the selection of membership functions more natural. For the composite variable *AB* (α-β), the normalization process previously described means that the units for this variable are already in normalized form. The units *dDeg* used in the knowledge base, are intended to represent *difference-degrees*, a somewhat arbitrary unit of measure.

4.6 Membership Functions

The placing and overlap of the membership functions happens to be one of the more critical design challenges. Unfortunately, it is also one of the most subjective and we rely heavily on *expert* knowledge and experience to establish these parameters. Too much overlap can blur the distinction between the set values and requires more membership functions to control the same range. Too little overlap can produce excessive control swings in the transition area leading to erratic behaviour. A general rule-of-thumb used in many fuzzy designs is that membership functions should overlap about 25%. In general, intuition suggests that the width of the membership functions around zero and nearest this middle point should be narrower than those at the extremes. This gives more precise control action where it is needed - near the steady state zero portion of the rule matrix. In other words, the most critical control actions occur as we are approaching near-perfect alignment close to the dock. Some experimentation in the design of these values was done to *fine-tune* the control performance, although the actual values used are not critical for successful docking of the trailer. The parameter values used in the final design are shown in the knowledge base listing in **Appendix A.** The shape and overlap of the membership functions for each input variable are shown as originally approximated in **Figure 15.**

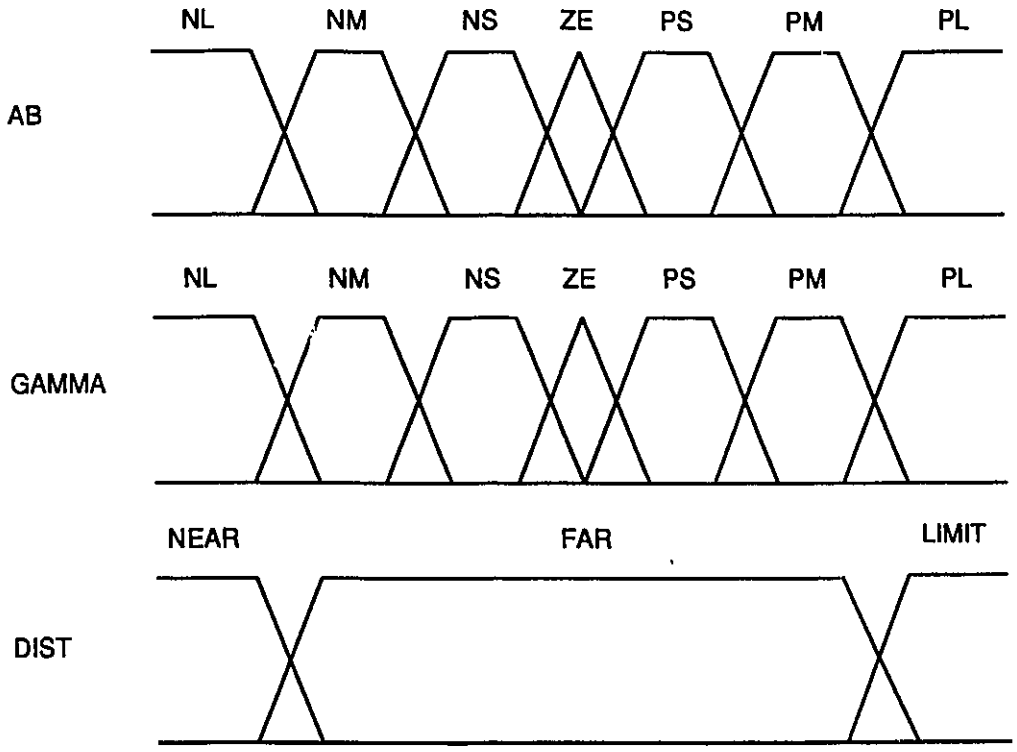


Figure 15 - Fuzzy Input Membership Functions

The output membership functions, being singletons in this application, are relatively straightforward to design. The direction variable is a binary output variable which is very simply used by the motor control interrupt routine to drive the truck in the appropriate direction. Its two singleton points are not critical and were arbitrarily set to values of 64 and 192. In fact, forward can be any point in the range of 0 to 127 and reverse can be any point in the range of 128 to 255. As a two's-complement number, this makes forward a positive number and reverse a negative number, which is very easy to check for in assembler code. In actual fact, there is nothing *fuzzy* at all about this variable - we either go forward or reverse. This is a good example of the *extension principle* referred to in the overview - in the limit, fuzzy sets can be *crisp*.

The vehicle speed is obviously not a critical parameter. It could just as easily have been kept constant and a simple override used on approaching the dock to slow down and stop the vehicle. This makes the speed variable somewhat redundant; however, it was controlled as a fuzzy variable because of the ease of experimentation with different values.

The actual values shown in **Figure 16** were established experimentally to allow safe speed throughout the range of the tether. An override in the user code checks the distance value after each call to the fuzzy engine and simply shuts down the drive motor and halts the MCU if the distance is zero (dock position) or > 36" (end of tether).

The steering variable is the only output parameter of any real consequence; hence its membership function points are critical. Again, the distance between the singleton points is used to establish finer control near the center-point. The outermost singleton points were established by experimentation at the practical limit of the steering servo, both hard-left and hard-right.

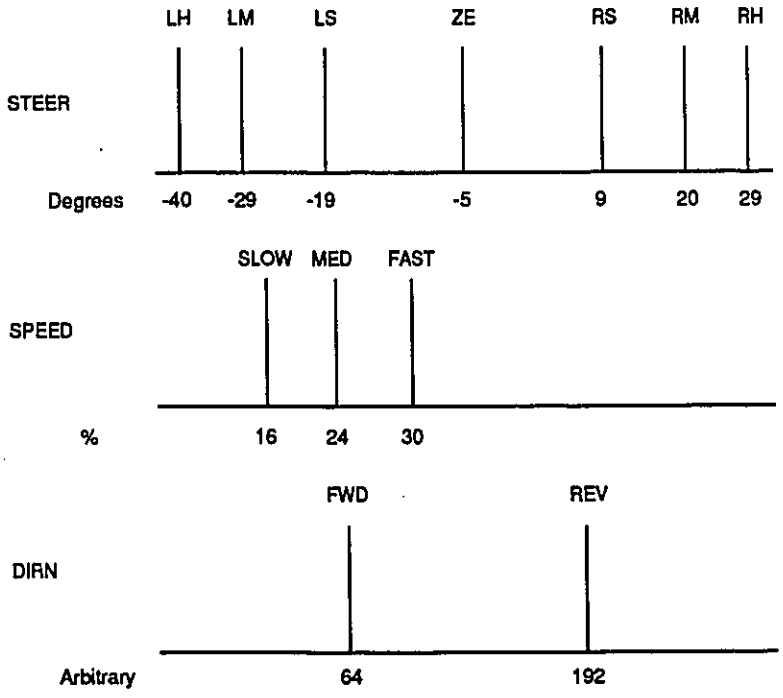


Figure 16 - Fuzzy Output Membership Functions

4.7 The Rule Base

The design of the rule base is also subjective and, in practice, can often be very difficult to design. Again, we rely heavily on expert knowledge and intuition. There are alternative methods using, for instance, neural networks to adaptively generate the rules using *product-space clustering* mentioned earlier. This technique is complex as it is difficult to generate the training data. Two techniques could be used.

Allowing several human operators (preferably somewhat adept at truck and trailer backing) to back up the truck, we could collect data from the position sensors and output controls for a large number of backup trials. Another method would be to build a software simulator of the system and use Nguyen andWidrow's technique to generate training data. For instructional purposes, this might be interesting future research. In actual fact, the rules in this case are not difficult to determine and appear to be not critical for successful docking behaviour.

As represented here, the problem has been converted from a truck backer upper to a truck docking problem. To some degree, allowing both forward and reverse motion simplifies the backing up task. Quite often, inexperienced drivers will succeed in backing up a trailer correctly by repeatedly going forward and backward. However, it does introduce another interesting control problem, that of how and when to switch from forward mode to backup mode.

In fact, in order to successfully switch from forward to reverse, there must be some hysteresis in the controller rule base; else we could easily get into an endless sequence of "jiggles" as the controller keeps changing its mind. Fortunately, there is a simple way of introducing hysteresis using the concept of a *dead-band* within the rule-base table.

STEER/DIRN RULES

AB \ GAMMA	NL	NM	NS	ZE	PS	PM	PL
NL	LH/F	LH/F	LH/F	LH/F	LM/F	LS/F	ZE/F
NM	LH/F	LH	LH	LM	LS	ZE	RS/F
NS	LH/F	LH	LM/R	LS/R	ZE/R	RS	RM/F
ZE	LH/F	LM	LS/R	ZE/R	RS/R	RM	RH/F
PS	LM/F	LS	ZE/R	RS/R	RM/R	RH	RH/F
PM	LS/F	ZE	RS	RM	RH	RH	RH/F
PL	ZE/F	RS/F	RM/F	RH/F	RH/F	RH/F	RH/F

Figure 17 - The Original Steering/Direction Rule Base

The easiest way to design rules is to use a rule matrix (see **Figure 17**) and begin by filling in the steady-state zero in the center. This indicates no control action is required when we are approaching the dock with perfect alignment (*AB* and γ are zero). Now we tackle the problem of how to design rules to control direction of travel. Earlier we indicated that as long as we are in a position from which we can successfully dock, we will continue to reverse and assume that we will approach the dock. That is, we will only switch to forward if we perceive that we are in trouble.

Because of the dynamics of this particular truck and trailer, there are two situations which will cause a problem. If γ is greater than about 75 or 80 degrees left or right, the truck and trailer are effectively *jack-knifed* and cannot recover in reverse at *any* steering angle. In addition, if α is very much greater than β or vice versa, then the truck cannot align properly with the dock because of the limitations of the tether (three feet) and by virtue of the fact that none of the potentiometers can travel in a full circle. In either of these situations, we must go forward and re-align properly before backing up again (see **Figure 18**). This is precisely what a human driver does in these situations.

Examining the rule base in **Figure 17**, we see that the outer ring of the matrix corresponds to these extreme situations and therefore, we will switch to forward motion whenever the system enters this area. The inner ring corresponds to alignment positions where the angle of the cab and trailer, γ is small *and* α is very close to β (*AB* is small). Under these conditions, the trailer is definitely under control and the backing up task can almost certainly be completed successfully. Therefore, we will ensure that we are in reverse in this case.

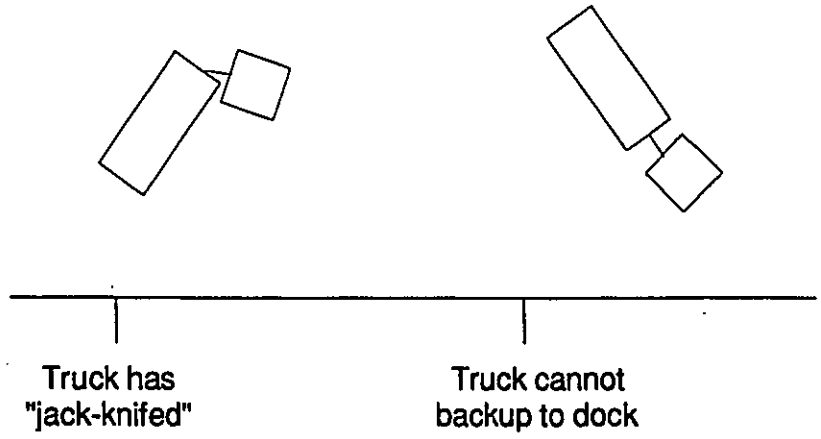


Figure 18 - Some Awkward Initial Positions

In fact, the in-between case (shaded area of the matrix) where the aforementioned angles are moderate can also be usually controlled successfully in reverse; however, we do not switch to either forward or reverse when in this situation. That is, the truck maintains its current direction until it enters either the inner or outer ring. In this way, we allow the truck to go forward until alignment is considerably improved before switching to reverse. In addition, we also do not give up too easily when backing up to give the system a chance of regaining control.

Next consider the design of the steering rules. Because of the symmetry of the problem, only the upper or lower diagonal of this matrix needs to be designed. It is only necessary to remember that our goal is to maintain α equal to β. To repeat our discussion earlier, if α is larger than β, then we must have the cab to the left of center *to the degree* that α is larger than β. Whether we need to steer left or right depends on where the cab presently is, i.e., if the cab is already to the left we may not have to alter the steering at all. On the other hand if the cab is to the right and we are in reverse, then we will have to steer right (going forward, we may not have to alter steering to increase β).

Using these simple heuristic notions of the dynamics of the truck and trailer, we fill in the rest of the table and give the system a test. In this case, even the first attempt succeeded - successful docking behaviour is not that sensitive to the selection of rules. However, by careful examination of the system's behaviour, it is possible again to fine-tune the rule base to smoothen the control process and possibly eliminate unnecessary switching of direction. The final rule base exhibiting very smooth docking behaviour is shown in **Figure 19.** Note the slightly skewed diagonal symmetry present in this matrix.

STEER/DIRN RULES

AB \ GAMMA	NL	NM	NS	ZE	PS	PM	PL
NL	LH/F	LH/F	LH/F	LM/F	LS/F	RS/F	RM/F
NM	LH/F	LH	LH	LM	ZE	RM	RH/F
NS	LH/F	LH	LM/R	LS/R	RS/R	RM	RH/F
ZE	LH/F	LH	LS/R	ZE/R	RS/R	RH	RH/F
PS	LH/F	LM	LS/R	RS/R	RM/R	RH	RH/F
PM	LH/F	LM	ZE	RM	RH	RH	RH/F
PL	LM/F	LS/F	RS/F	RM/F	RH/F	RH/F	RH/F

Figure 19 - Final Steering Rule Base

The rule base for the *SPEED* output value is trivial, since we only control speed by distance *d* from the dock. The rule matrix for this control function is shown in **Figure** 20.

SPEED RULES

DISTANCE		
NEAR	FAR	LIMIT
SLOW	FAST	MEDIUM

Figure 20 - The Speed Rule Base

CHAPTER 5

RESULTS AND CONCLUSIONS

5.1 Experimental Results

Early in the research, an attempt was made to design a traditional controller using a simple hard-left or hard-right steering signal based on a functional analysis of inputs. The idea was to have a rudimentary controller for comparison with the ultimate fuzzy controller. This design process was frustrating and eventually abandoned. As suggested by other researchers' experience, the problem is difficult to solve by conventional control techniques, even for motion in only one direction.

On the other hand, even the first rudimentary knowledge base tested performed reasonably well in approaching the dock from most positions, although final alignment was poor. Having built and tested the various sub-systems, experimentation is essentially reduced to modifications to the fuzzy membership functions and rule base. The current knowledge base has been tested from various initial configurations and exhibits appropriate docking behaviour at speeds of approximately 10 to 20 cm/sec. Since a complete cycle of the fuzzy inference engine typically requires less than 1 ms, the truck can approach the dock at a robust speed, slowing down only as it nears the dock.

Final docking alignment is occasionally poor because of the limitations of the steering wheels and friction in the position sensing potentiometers, i.e., the wire occasionally fails to keep the potentiometer arms strictly in line. In severe cases, the truck simply switches direction, drives forward until alignment has improved and is then able to complete the backing up task satisfactorily.

The mechanical positioning system is sensitive and requires frequent re-calibration but performs well from positions within its range of 3 feet. The optical system described earlier has undergone initial testing to prove its feasibility and should be operational shortly. It is anticipated that this laser-based position sensing system will overcome the limitations of the present mechanical system and allow operation from greater ranges (in the order of 10 to 12 feet).

Shown in **Figure 21** through **Figure 25** are representations of typical trajectories followed by the truck as it approaches the dock from various initial configurations. In **Figure 21**, the truck's initial position allows it to dock with excellent alignment using only small steering control actions. No forward motion is required. Even in the next situation (**Figure 22**) where initial alignment is not good, the truck can approach the dock without the need to shift into forward. In this case, the final alignment is poorer.

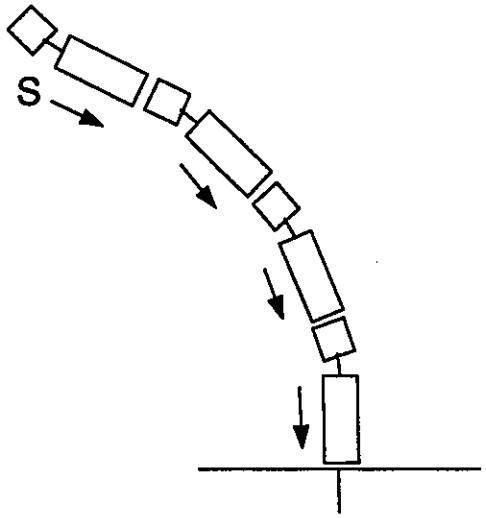


Figure 21 - Good Initial Alignment
The truck backs up immediately to the dock

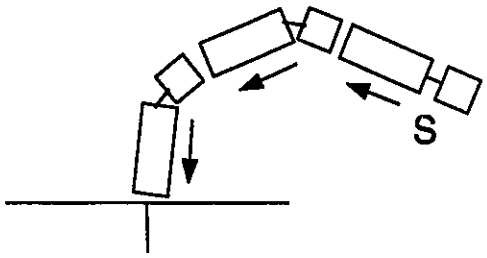


Figure 22 - Mediocre Alignment
Immediate backup with poor final alignment

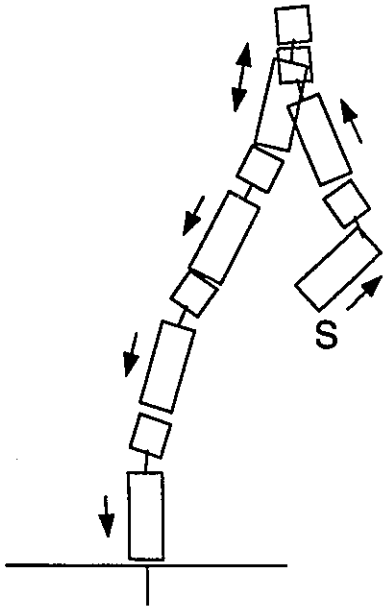


Figure 23 - Cab is Jack-knifed
Truck goes forward to re-align before docking

In **Figure 23**, the initial jack-knife position of the truck and trailer causes it to immediately switch to the forward position. It continues to move forward only until the cab has straightened somewhat and then switches to reverse and is able to dock successfully. In this case, the alignment of the trailer is initially good and is not a factor in the decision to switch direction.

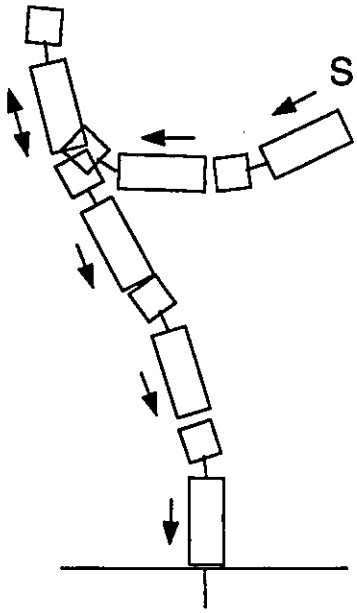


Figure 24 - An Awkward Trailer Angle
Truck again must go forward to re-align

Conversely in **Figure 24**, the cab is initially aligned well with the trailer but the trailer is at a very awkward angle with the dock. Again, the truck immediately moves forward until alignment improves before backing to the dock.

Finally, in **Figure 25**, the truck first attempts to back up to the dock but cannot succeed because of its initial position. It gets into an awkward position, switches to forward and tries again to back up to the dock. Sometimes, as in this case, the backup task does not immediately succeed and the truck must switch to forward again to improve alignment. With earlier membership functions, this behaviour of repeatedly switching between forward and reverse occurred often.

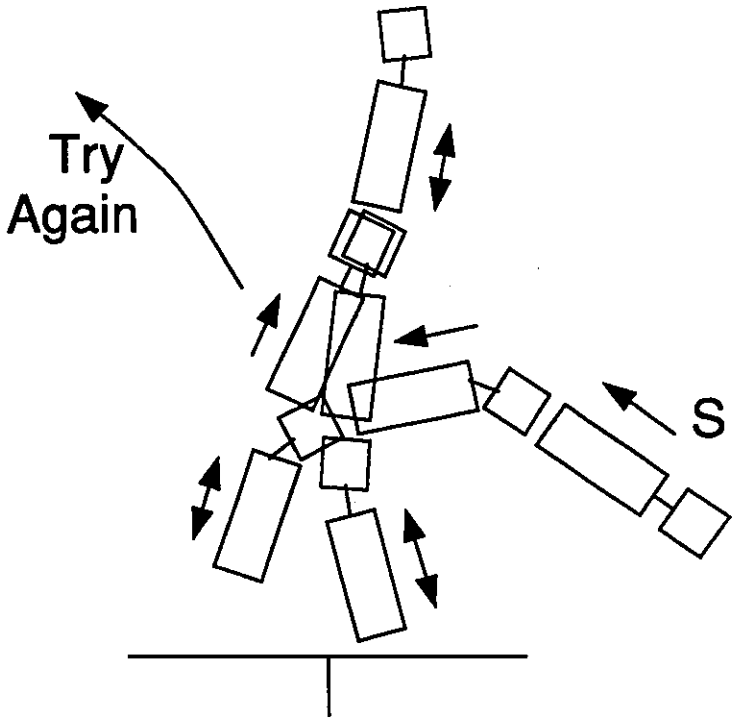


Figure 25 - An Interesting Attempt
The truck must switch directions several times to complete the backing-up task

Fine tuning of the membership functions was required to produce good docking behaviour. This fine tuning process is intuitive in the case of the truck docking problem because of the small number of variables and the simple rule base but, in larger problems, could pose difficulties.

The final input membership functions shown in **Figure 26** are the result of this tuning process. Notice how the non-linearity of the control problem is reflected in the membership functions. In particular, the membership functions of *ZE* (zero) and *NS* and PS (*Negative Small* and *Positive Small*) for variables *GAMMA* and *AB* are much narrower than those for the extreme positions. This *tightening up* of the functions near the steady-state zero is necessary to produce adequate control behaviour as the truck makes final corrections near the end of the docking task.

In addition, the extra wide medium regions (*NM* and *PM*) are necessary to discourage repeated switching of direction. This effectively enlarges the *dead-zone* area of the rule table and ensures that forward motion continues long enough to sufficiently improve alignment.

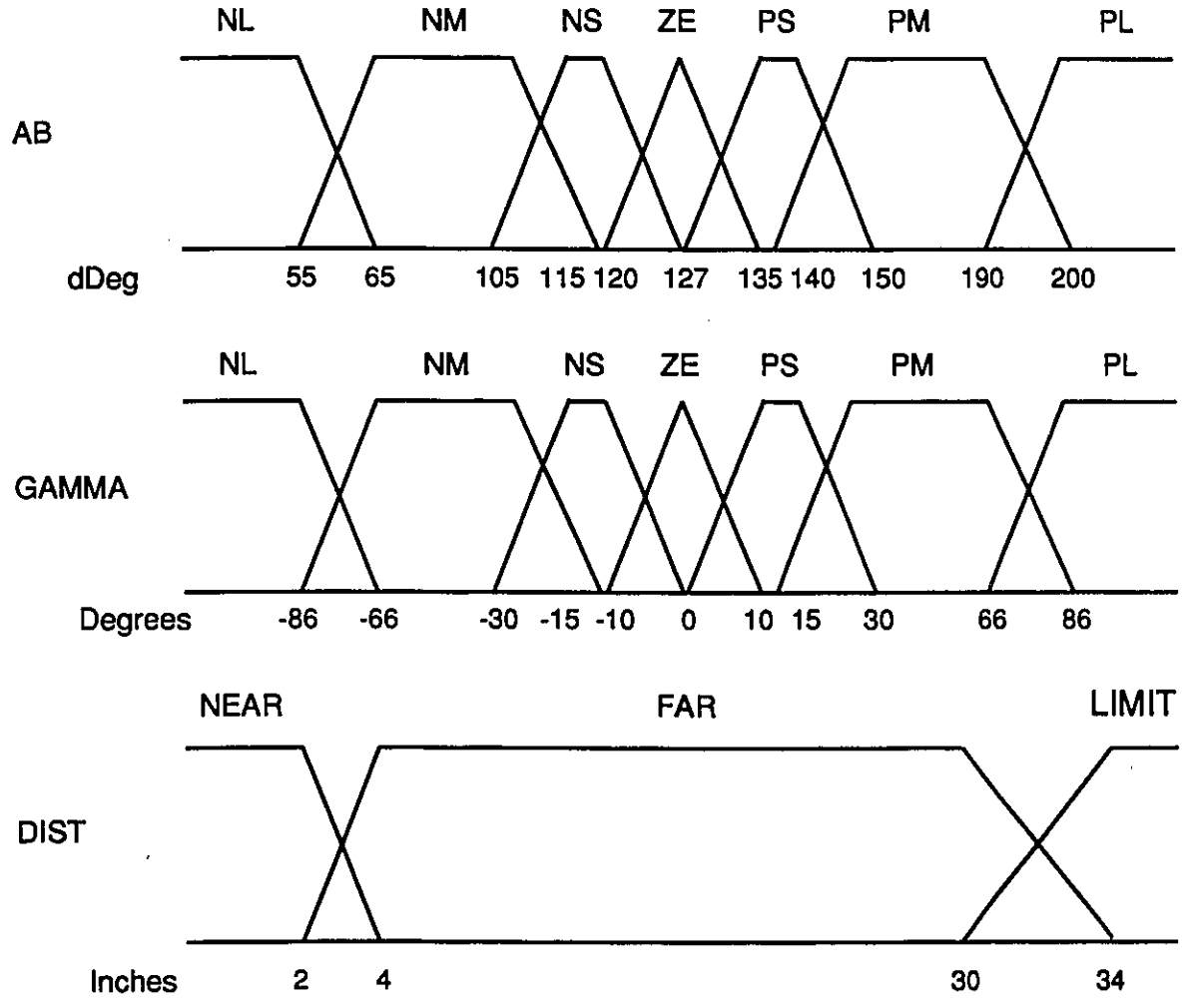


Figure 26 - The Final Input Membership Functions

5.2 Conclusions

Under the control of the fuzzy logic membership functions and rule base, the fuzzy truck exhibits reasonably good docking performance, in some cases acting very much as if driven by a human operator. This should not be surprising since the fuzzy system parameters were determined primarily from expert human experience. The limitations in performance are primarily a result of the limited steering control available with this vehicle and the inherent error in the mechanical position sensing system due to friction.

The experiments described have generally been regarded as a successful demonstration of autonomous vehicle behaviour using fuzzy logic. The truck was demonstrated at the most recent North American Fuzzy Information Processing Society meeting (NAFIPS-93) held in Allentown, PA in August of 1993 and generated much interest. In addition, a paper describing this research has been accepted for publishing in a book entitled "Fuzzy Logic and Intelligent Systems" to be published later this year by Kluwer Academic Publishers.

The fuzzy truck is more than just a demonstration of fuzzy logic, however. It can be used as a test bed for alternative control methods, such as neural networks, and alternative real-world sensor systems. In addition, it can be used for experimenting with other autonomous behaviours such as collision avoidance and adaptive learning.

The fuzzy truck has obvious educational utility in exploring real-world AI applications. This project covers a broad range of engineering discipline including mechanical, electrical and control engineering not to mention fuzzy systems. The cost of the project is less than $250 provided the builder is willing to be somewhat innovative in the use of surplus and spare parts. It would make an excellent group science fair project as it gives an interesting visual demonstration of the power of fuzzy logic in producing autonomous behaviour.

CHAPTER 6

FUTURE RESEARCH

6.1 Mobile Robots

Experience gained in these preliminary experiments will be useful for planned future research in the area of autonomous control of mobile robots. The existing fuzzy controller and optical sensor system could, with very little modification, be used to control any wheeled vehicle, with major changes likely required for the rule base only. In particular, it appears that fuzzy logic may be one of the only methodologies capable of satisfactorily controlling articulated vehicles.

6.2 Collision Avoidance

The most obvious extension to the current system would be to include collision avoidance behaviour. In the simple case presented here of truck and trailer docking, this would allow more interesting docking behaviour closer in reality to actual trailer docking situations. This is not difficult to achieve conceptually using the optical sensor system. Retro-reflective materials of differing patterns would distinguish an obstacle from the dock. The most difficult problem is to determine relative distance to an obstacle. Given that the truck is moving, this could be accomplished through triangulation; however, it seems more likely that a more accurate ranging device would be required. It is anticipated that the collision avoidance behaviour could be provided by additional rules acting on this sensor data, i.e.:

IF obstacle is near-left AND direction is forward THEN steer hard right

6.3 Space Rendezvous and Docking

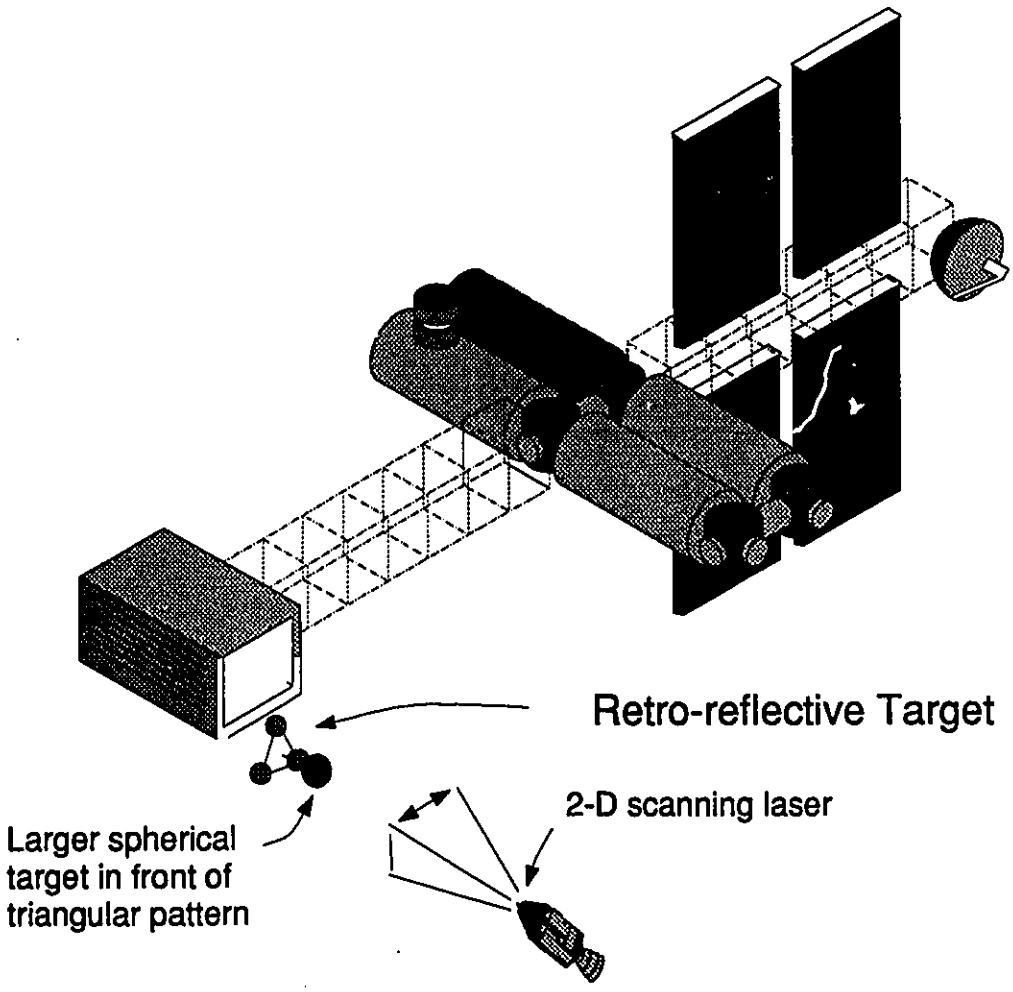


Figure 27 - A 3-D Docking Target

If a two-dimensional scanning laser were incorporated in the position sensing system, the fuzzy controller could readily be extended to operate in three dimensions for autonomous space rendezvous and docking applications (see **Figure 27**). This is an important area of research for control of planetary surface rovers and deep space unmanned vehicles where telepresence control (remote human operator) is precluded because of the long signal transit time.

In a recent paper presented at NAFIPS [19], the authors propose just such a system using fuzzy logic to control spacecraft thrusters. The concept has been tested at Marshall Space Flight Center using a standard video camera and a Remote Manipulator System (RMS) docking target.

REFERENCES

1. P.J. McKerrow, "Introduction to Robotics", Addison-Wesley, 1991.

2. M. Brady, "Artificial Intelligence and Robotics", Artificial Intelligence, vol. 26, pp. 79-121, 1985.

3. "Missions, Technologies and Design of Planetary Mobile Vehicles", CNES Symposium, Toulouse, France, Sept. 1992.

4. L.A. Zadeh, "Fuzzy Sets", Information and Control, vol. 8, pp. 338-353, 1965.

5. A. Kaufmann and M.M. Gupta, "Introduction to Fuzzy Arithmetic", Reinhold, New York, 1985.

6. W. Bandler and L.J. Kohout, "Fuzzy Power Sets and Fuzzy Implication Operators", Fuzzy Sets and Systems, vol. 4, pp. 13-30, 1980.

7. Dubois, Didier and Prade, "A Class of Fuzzy Measures Based on Triangle Inequalities", Int. J. Gen. Sys. vol. 8.

8. L. Zadeh, "Fuzzy Sets", Inf. and Control, vol. 8, pp. 338-353, 1965.

9. L. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems", IEEE Trans. on Sys., Man and Cyb., vol. 3, 1973.

10. L. Zadeh, "The Calculus of Fuzzy Restrictions", in Fuzzy Sets and Applications to Cognitive and Decision Making Processes, edited by L. Zadeh et. al., Academic Press, New York, 1975.

11. E.H. Mamdani, Applications of Fuzzy Algorithms for Control of Simple Dynamic Plant, Proc. IEEE, vol. 121, pp. 1585-1588, 1976.

12. M. Sugeno et. al., "Helicopter Flight Control Based on Fuzzy Logic", in Fuzzy Engineering toward Human Friendly Systems, edited by T. Terano, M. Sugeno, M. Mukaidono and K. Shigemasu, IOS Press, Burke, VA, 1992.

13. "NASA Eyes Range of Fuzzy Control Ideas in Space", Computer Design, April, 1992.

14. D. Nguyen and B. Widrow "The Truck Backer-Upper: An Example of Self-Learning in Neural Networks", Proceedings of International Joint Conference on Neural Networks (IJCNN-89), vol. II, pp. 357-363, June 1989.

15. B. Kosko, "Comparison of fuzzy and neural control systems", in Neural Networks and Fuzzy Systems, pp. 339-361 (1992)

16. E.S. Plumer "Neural Network Structure for Navigation using Potential Fields" in Proceedings of International Joint Conference on Neural Networks, (IJCNN-92), vol. I, pp. 327-332, July 1992.

17. J.R. Koza "A Genetic Approach to the Truck Backer Upper Problem and the Inter-Twined Spiral Problem", in Proceedings of International Joint Conference on Neural Networks (IJCNN-92), vol. IV, pp. 310-318, July 1992.

18. MC68HC11 Reference Manual, Motorola Reference #M68HC11RM/AD, Prentice Hall, NJ, 1989.

19. V. Gopalan, A. Homaifar, B. Sayyarrodsari, "Fuzzy Hierarchial Controller for Autonomous Rendezvous and Docking Problem", Proceedings of 12th Annual Meeting of NAFIPS, August, 1993.

APPENDIX A – KNOWLEDGE BASE LISTING

KB_NEW.PRN: INPUTS

Input #1

INPUT NAME	Minimum Value	Maximum Value	Units
AB	0	255	dDeg

MEMBERSHIP FUNCTION	P1	P2	P3	P4
NL	0	0	55	65
NM	55	65	105	115
NS	105	115	120	127
ZE	120	127	128	135
PS	128	135	140	150
PM	140	150	190	200
PL	190	200	255	255
~	0	0	0	0

Input #2

INPUT NAME	Minimum Value	Maximum Value	Units
Gamma	-144	144	Deg

MEMBERSHIP FUNCTION	P1	P2	P3	P4
NL	-144	-144	-86	-66
NM	-86	-66	-30	-14
NS	-30	-15	-8	1
ZE	-10	1	1	11
PS	1	10	15	30
PM	15	30	66	86
PL	66	86	144	144
~	0	0	0	0

Input #3

INPUT NAME	Minimum Value	Maximum Value	Units
Dist	0	36	Inches

MEMBERSHIP FUNCTION	P1	P2	P3	P4
NEAR	0	0	2	4
FAR	2	4	30	34
LMT	30	34	36	36
~	0	0	0	0
~	0	0	0	0
~	0	0	0	0
~	0	0	0	0
~	0	0	0	0

APPENDIX A – KNOWLEDGE BASE LISTING

KB_NEW.PRN: OUTPUTS

Output #1

OUTPUT NAME	Minimum Value	Maximum Value	Units
Steer	-45	45	angle

MEMBERSHIP FUNCTION P1

LH	-40
LM	-29
LS	-19
ZE	-5
RS	9
RM	20
RH	29
~	0

Output #2

OUTPUT NAME	Minimum Value	Maximum Value	Units
Speed	0	100	%

MEMBERSHIP FUNCTION P1

SLOW	16
MED	24
FAST	30
~	0
~	0
~	0
~	0
~	0

Output #3

OUTPUT NAME	Minimum Value	Maximum Value	Units
Dirn	0	255	norm.

MEMBERSHIP FUNCTION P1

FWD	64
REV	192
~	0
~	0
~	0
~	0
~	0
~	0

APPENDIX A – KNOWLEDGE BASE LISTING

KB_NEW.PRN: RULES

```
Rule # 1:
IF Dist IS LMT
    THEN Speed IS MED

Rule # 2:
IF Dist IS FAR
    THEN Speed IS FAST

Rule # 3:
IF Dist IS NEAR
    THEN Speed IS SLOW

Rule # 4:
IF AB IS NL
      AND Gamma IS NL
    THEN Steer IS LH
      AND Dirn IS FWD

Rule # 5:
IF AB IS NL
      AND Gamma IS NM
    THEN Steer IS LH
      AND Dirn IS FWD

Rule # 6:
IF AB IS NL
      AND Gamma IS NS
    THEN Steer IS LH
      AND Dirn IS FWD

Rule # 7:
IF AB IS NL
      AND Gamma IS ZE
    THEN Steer IS LM
      AND Dirn IS FWD

Rule # 8:
IF AB IS NL
      AND Gamma IS PS
    THEN Steer IS LS
      AND Dirn IS FWD

Rule # 9:
IF AB IS NL
      AND Gamma IS PM
    THEN Steer IS RS
      AND Dirn IS FWD

Rule #10:
IF AB IS NL
      AND Gamma IS PL
    THEN Steer IS RM
      AND Dirn IS FWD

Rule #11:
IF AB IS NM
      AND Gamma IS NL
    THEN Steer IS LH
      AND Dirn IS FWD

Rule #12:
IF AB IS NM
      AND Gamma IS NM
    THEN Steer IS LH

Rule #13:
IF AB IS NM
      AND Gamma IS NS
    THEN Steer IS LH

Rule #14:
IF AB IS NM
      AND Gamma IS ZE
    THEN Steer IS LM

Rule #15:
IF AB IS NM
      AND Gamma IS PS
    THEN Steer IS ZE

Rule #16:
IF AB IS NM
      AND Gamma IS PM
    THEN Steer IS RM

Rule #17:
IF AB IS NM
      AND Gamma IS PL
    THEN Steer IS RH
      AND Dirn IS FWD

Rule #18:
IF AB IS NS
      AND Gamma IS NL
    THEN Steer IS LH
      AND Dirn IS FWD

Rule #19:
IF AB IS NS
      AND Gamma IS NM
    THEN Steer IS LH

Rule #20:
IF AB IS NS
      AND Gamma IS NS
  THEN Steer IS LM
      AND Dirn IS REV
```

APPENDIX A – KNOWLEDGE BASE LISTING

KB_NEW.PRN: RULES

```
Rule #21:
IF AB IS NS
      AND Gamma IS ZE
    THEN Steer IS LS
      AND Dirn IS REV

Rule #22:
IF AB IS NS
      AND Gamma IS PS
    THEN Steer IS RS
      AND Dirn IS REV

Rule #23:
IF AB IS NS
      AND Gamma IS PM
    THEN Steer IS RM

Rule #24:
IF AB IS NS
      AND Gamma IS PL
    THEN Steer IS RH
      AND Dirn IS FWD

Rule #25:
IF AB IS ZE
      AND Gamma IS NL
    THEN Steer IS LH
      AND Dirn IS FWD

Rule #26:
IF AB IS ZE
      AND Gamma IS NM
    THEN Steer IS LH

Rule #27:
IF AB IS ZE
      AND Gamma IS NS
    THEN Steer IS LS
      AND Dirn IS REV

Rule #28:
IF AB IS ZE
      AND Gamma IS ZE
    THEN Steer IS ZE
      AND Dirn IS REV

Rule #29:
IF AB IS ZE
      AND Gamma IS PS
    THEN Steer IS RS
      AND Dirn IS REV

Rule #30:
IF AB IS ZE
      AND Gamma IS PM
    THEN Steer IS RH

Rule #31:
IF AB IS ZE
      AND Gamma IS PL
    THEN Steer IS RH
      AND Dirn IS FWD

Rule #32:
IF AB IS PS
      AND Gamma IS NL
    THEN Steer IS LH
      AND Dirn IS FWD

Rule #33:
IF AB IS PS
      AND Gamma IS NM
    THEN Steer IS LM

Rule #34:
IF AB IS PS
      AND Gamma IS NS
    THEN Steer IS LS
      AND Dirn IS REV

Rule #35:
IF AB IS PS
      AND Gamma IS ZE
    THEN Steer IS RS
      AND Dirn IS REV

Rule #36:
IF AB IS PS
      AND Gamma IS PS
    THEN Steer IS RM
      AND Dirn IS REV

Rule #37:
IF AB IS PS
      AND Gamma IS PM
    THEN Steer IS RH

Rule #38:
IF AB IS PS
      AND Gamma IS PL
    THEN Steer IS RH
      AND Dirn IS FWD

Rule #39:
IF AB IS PM
      AND Gamma IS NL
    THEN Steer IS LH
      AND Dirn IS FWD

Rule #40:
IF AB IS PM
      AND Gamma IS NM
    THEN Steer IS LM
```

APPENDIX A – KNOWLEDGE BASE LISTING

KB_NEW.PRN: RULES

```
Rule #41:
IF AB IS PM
      AND Gamma IS NS
    THEN Steer IS ZE

Rule #42:
IF AB IS PM
      AND Gamma IS ZE
    THEN Steer IS RM

Rule #43:
IF AB IS PM
      AND Gamma IS PS
    THEN Steer IS RH

Rule #44:
IF AB IS PM
      AND Gamma IS PM
    THEN Steer IS RH

Rule #45:
IF AB IS PM
      AND Gamma IS PL
    THEN Steer IS RH
      AND Dirn IS FWD

Rule #46:
IF AB IS PL
      AND Gamma IS NL
    THEN Steer IS LM
      AND Dirn IS FWD

Rule #47:
IF AB IS PL
      AND Gamma IS NM
    THEN Steer IS LS
      AND Dirn IS FWD

Rule #48:
IF AB IS PL
      AND Gamma IS NS
    THEN Steer IS RS
      AND Dirn IS FWD

Rule #49:
IF AB IS PL
      AND Gamma IS ZE
    THEN Steer IS RM
      AND Dirn IS FWD

Rule #50:
IF AB IS PL
      AND Gamma IS PS
    THEN Steer IS RH
      AND Dirn IS FWD

Rule #51:
IF AB IS PL
      AND Gamma IS PM
    THEN Steer IS RH
      AND Dirn IS FWD

Rule #52:
IF AB IS PL
      AND Gamma IS PL
    THEN Steer IS RH
      AND Dirn IS FWD
```

APPENDIX B – KNOWLEDGE BASE ASSEMBLY CODE

```
0001                                *       Knowledge Base Generator Version V2.22
0002                                *       Author: John Dumas & Jason Spielman.
0003                                *       Copyright Motorola 1991,1992
0004                                *       File Name = KB_NEW.ASM
0005                                *       Created for FUZZ3B.ASM Engine only.
0006 1800                                     ORG     $100
0007 1800                           INPUT_MFS EQU     *               ; Input Membership Functions.
0008 1800                           IN0MF     EQU     *               ;              AB
0009 1800 00 00 37 1a                         FCB     $00,$00,$37,$1A ;              NL
0010 1804 37 1a 69 1a                         FCB     $37,$1A,$69,$1A ;              NM
0011 1808 69 1a 78 24                         FCB     $69,$1A,$78,$24 ;              NS
0012 180c 78 24 80 24                         FCB     $78,$24,$80,$24 ;              ZE
0013 1810 80 24 8c 1a                         FCB     $80,$24,$8C,$1A ;              PS
0014 1814 8c 1a be 1a                         FCB     $8C,$1A,$BE,$1A ;              PM
0015 1818 be 1a ff 00                         FCB     $BE,$1A,$FF,$00 ;              PL
0016 181c 00 00 00 00                         FCB     $00,$00,$00,$00 ;               -
0017 1820                           IN1MF     EQU     *               ;           Gamma
0018 1820 00 00 33 0e                         FCB     $00,$00,$33,$0E ;              NL
0019 1824 33 0e 65 12                         FCB     $33,$0E,$65,$12 ;              NM
0020 1828 65 13 78 20                         FCB     $65,$13,$78,$20 ;              NS
0021 182c 77 1b 80 1d                         FCB     $77,$1B,$80,$1D ;              ZE
0022 1830 80 20 8d 13                         FCB     $80,$20,$8D,$13 ;              PS
0023 1834 8d 13 ba 0e                         FCB     $8D,$13,$BA,$0E ;              PM
0024 1838 ba 0e ff 00                         FCB     $BA,$0E,$FF,$00 ;              PL
0025 183c 00 00 00 00                         FCB     $00,$00,$00,$00 ;               -
0026 1840                           IN2MF     EQU     *               ;            Dist
0027 1840 00 00 0e 10                         FCB     $00,$00,$0E,$10 ;            NEAR
0028 1844 0e 10 d5 09                         FCB     $0E,$10,$D5,$09 ;             FAR
0029 1848 d5 09 ff 00                         FCB     $D5,$09,$FF,$00 ;             LMT
0030 184c 00 00 00 00                         FCB     $00,$00,$00,$00 ;               -
0031 1850 00 00 00 00                         FCB     $00,$00,$00,$00 ;               -
0032 1854 00 00 00 00                         FCB     $00,$00,$00,$00 ;               -
0033 1858 00 00 00 00                         FCB     $00,$00,$00,$00 ;               -
0034 185c 00 00 00 00                         FCB     $00,$00,$00,$00 ;               -
0035 1860                           SGLTN_POS EQU     *               ; Output Membership Functions.
0036 1860                           OUT0MF    EQU     *               ;           Steer
0037 1860 0e                                  FCB     $0E             ;              LH
0038 1861 2d                                  FCB     $2D             ;              LM
0039 1862 49                                  FCB     $49             ;              LS
0040 1863 71                                  FCB     $71             ;              ZE
0041 1864 99                                  FCB     $99             ;              RS
0042 1865 b8                                  FCB     $B8             ;              RM
0043 1866 d1                                  FCB     $D1             ;              RH
0044 1867 7f                                  FCB     $7F             ;               -
0045 1868                           OUT1MF    EQU     *               ;           Speed
0046 1868 28                                  FCB     $28             ;            SLOW
0047 1869 3d                                  FCB     $3D             ;             MED
0048 186a 4c                                  FCB     $4C             ;            FAST
0049 186b 00                                  FCB     $00             ;               -
0050 186c 00                                  FCB     $00             ;               -
0051 186d 00                                  FCB     $00             ;               -
```

APPENDIX B – KNOWLEDGE BASE ASSEMBLY CODE

```
0052 186e 00                        FCB     $00             ;               -
0053 186f 00                        FCB     $00             ;               -
0054 1870           OUT2MF          EQU     *               ;               Dirn
0055 1870 40                        FCB     $40             ;               FWD
0056 1871 c0                        FCB     $C0             ;               REV
0057 1872 00                        FCB     $00             ;               -
0058 1873 00                        FCB     $00             ;               -
0059 1874 00                        FCB     $00             ;               -
0060 1875 00                        FCB     $00             ;               -
0061 1876 00                        FCB     $00             ;               -
0062 1877 00                        FCB     $00             ;               -
0063 1878           RULE_START      EQU     *               ; Rules follow:
0064 1878 12                        FCB     $12
0065 1879 89                        FCB     $89
0066 187a 11                        FCB     $11
0067 187b 8a                        FCB     $8A
0068 187c 10                        FCB     $10
0069 187d 88                        FCB     $88
0070 187e 00                        FCB     $00
0071 187f 08                        FCB     $08
0072 1880 80                        FCB     $80
0073 1881 90                        FCB     $90
0074 1882 00                        FCB     $00
0075 1883 09                        FCB     $09
0076 1884 80                        FCB     $80
0077 1885 90                        FCB     $90
0078 1886 00                        FCB     $00
0079 1887 0a                        FCB     $0A
0080 1888 80                        FCB     $80
0081 1889 90                        FCB     $90
0082 188a 00                        FCB     $00
0083 188b 0b                        FCB     $0B
0084 188c 81                        FCB     $81
0085 188d 90                        FCB     $90
0086 188e 00                        FCB     $00
0087 188f 0c                        FCB     $0C
0088 1890 82                        FCB     $82
0089 1891 90                        FCB     $90
0090 1892 00                        FCB     $00
0091 1893 0d                        FCB     $0D
0092 1894 84                        FCB     $84
0093 1895 90                        FCB     $90
0094 1896 00                        FCB     $00
0095 1897 0e                        FCB     $0E
0096 1898 85                        FCB     $85
0097 1899 90                        FCB     $90
0098 189a 01                        FCB     $01
0099 189b 08                        FCB     $08
0100 189c 80                        FCB     $80
0101 189d 90                        FCB     $90
0102 189e 01                        FCB     $01
```

APPENDIX B – KNOWLEDGE BASE ASSEMBLY CODE

```
0103 189f 09                FCB     $09
0104 18a0 80                FCB     $80
0105 18a1 01                FCB     $01
0106 18a2 0a                FCB     $0A
0107 18a3 80                FCB     $80
0108 18a4 01                FCB     $01
0109 18a5 0b                FCB     $0B
0110 18a6 81                FCB     $81
0111 18a7 01                FCB     $01
0112 18a8 0c                FCB     $0C
0113 18a9 83                FCB     $83
0114 18aa 01                FCB     $01
0115 18ab 0d                FCB     $0D
0116 18ac 85                FCB     $85
0117 18ad 01                FCB     $01
0118 18ae 0e                FCB     $0E
0119 18af 86                FCB     $86
0120 18b0 90                FCB     $90
0121 18b1 02                FCB     $02
0122 18b2 08                FCB     $08
0123 18b3 80                FCB     $80
0124 18b4 90                FCB     $90
0125 18b5 02                FCB     $02
0126 18b6 09                FCB     $09
0127 18b7 80                FCB     $80
0128 18b8 02                FCB     $02
0129 18b9 0a                FCB     $0A
0130 18ba 81                FCB     $81
0131 18bb 91                FCB     $91
0132 18bc 02                FCB     $02
0133 18bd 0b                FCB     $0B
0134 18be 82                FCB     $82
0135 18bf 91                FCB     $91
0136 18c0 02                FCB     $02
0137 18c1 0c                FCB     $0C
0138 18c2 84                FCB     $84
0139 18c3 91                FCB     $91
0140 18c4 02                FCB     $02
0141 18c5 0d                FCB     $0D
0142 18c6 85                FCB     $85
0143 18c7 02                FCB     $02
0144 18c8 0e                FCB     $0E
0145 18c9 86                FCB     $86
0146 18ca 90                FCB     $90
0147 18cb 03                FCB     $03
0148 18cc 08                FCB     $08
0149 18cd 80                FCB     $80
0150 18ce 90                FCB     $90
0151 18cf 03                FCB     $03
0152 18d0 09                FCB     $09
0153 18d1 80                FCB     $80
```

APPENDIX B – KNOWLEDGE BASE ASSEMBLY CODE

```
0154 18d2 03          FCB     $03
0155 18d3 0a          FCB     $0A
0156 18d4 82          FCB     $82
0157 18d5 91          FCB     $91
0158 18d6 03          FCB     $03
0159 18d7 0b          FCB     $0B
0160 18d8 83          FCB     $83
0161 18d9 91          FCB     $91
0162 18da 03          FCB     $03
0163 18db 0c          FCB     $0C
0164 18dc 84          FCB     $84
0165 18dd 91          FCB     $91
0166 18de 03          FCB     $03
0167 18df 0d          FCB     $0D
0168 18e0 86          FCB     $86
0169 18e1 03          FCB     $03
0170 18e2 0e          FCB     $0E
0171 18e3 86          FCB     $86
0172 18e4 90          FCB     $90
0173 18e5 04          FCB     $04
0174 18e6 08          FCB     $08
0175 18e7 80          FCB     $80
0176 18e8 90          FCB     $90
0177 18e9 04          FCB     $04
0178 18ea 09          FCB     $09
0179 18eb 81          FCB     $81
0180 18ec 04          FCB     $04
0181 18ed 0a          FCB     $0A
0182 18ee 82          FCB     $82
0183 18ef 91          FCB     $91
0184 18f0 04          FCB     $04
0185 18f1 0b          FCB     $0B
0186 18f2 84          FCB     $84
0187 18f3 91          FCB     $91
0188 18f4 04          FCB     $04
0189 18f5 0c          FCB     $0C
0190 18f6 85          FCB     $85
0191 18f7 91          FCB     $91
0192 18f8 04          FCB     $04
0193 18f9 0d          FCB     $0D
0194 18fa 86          FCB     $86
0195 18fb 04          FCB     $04
0196 18fc 0e          FCB     $0E
0197 18fd 86          FCB     $86
0198 18fe 90          FCB     $90
0199 18ff 05          FCB     $05
0200 1900 08          FCB     $08
0201 1901 80          FCB     $80
0202 1902 90          FCB     $90
0203 1903 05          FCB     $05
0204 1904 09          FCB     $09
```

APPENDIX B – KNOWLEDGE BASE ASSEMBLY CODE

```
0205 1905 81                          FCB     $81
0206 1906 05                          FCB     $05
0207 1907 0a                          FCB     $0A
0208 1908 83                          FCB     $83
0209 1909 05                          FCB     $05
0210 190a 0b                          FCB     $0B
0211 190b 85                          FCB     $85
0212 190c 05                          FCB     $05
0213 190d 0c                          FCB     $0C
0214 190e 86                          FCB     $86
0215 190f 05                          FCB     $05
0216 1910 0d                          FCB     $0D
0217 1911 86                          FCB     $86
0218 1912 05                          FCB     $05
0219 1913 0e                          FCB     $0E
0220 1914 86                          FCB     $86
0221 1915 90                          FCB     $90
0222 1916 06                          FCB     $06
0223 1917 08                          FCB     $08
0224 1918 81                          FCB     $81
0225 1919 90                          FCB     $90
0226 191a 06                          FCB     $06
0227 191b 09                          FCB     $09
0228 191c 82                          FCB     $82
0229 191d 90                          FCB     $90
0230 191e 06                          FCB     $06
0231 191f 0a                          FCB     $0A
0232 1920 84                          FCB     $84
0233 1921 90                          FCB     $90
0234 1922 06                          FCB     $06
0235 1923 0b                          FCB     $0B
0236 1924 85                          FCB     $85
0237 1925 90                          FCB     $90
0238 1926 06                          FCB     $06
0239 1927 0c                          FCB     $0C
0240 1928 86                          FCB     $86
0241 1929 90                          FCB     $90
0242 192a 06                          FCB     $06
0243 192b 0d                          FCB     $0D
0244 192c 86                          FCB     $86
0245 192d 90                          FCB     $90
0246 192e 06                          FCB     $06
0247 192f 0e                          FCB     $0E
0248 1930 86                          FCB     $86
0249 1931 90                          FCB     $90
0250 1932 ff          END_OF_RULE     FCB     $FF
0251 0003             NUMINP  EQU     $3
0252 0003             NUMOUT  EQU     $3
0253 1933 32 2e 32 32 DEFVER  FCC     '2.22'
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0001                    ********************************************************************************
0002                    * Fuzzy Truck Docker - GJE 94/04/04
0003                    *                    - register block moved to direct page
0004                    *                    - some code optimized to conserve space
0005                    ********************************************************************************
0006                    *
0007                    * Fuzzy mem funcs and rules assumed to be at $8000
0008                    *
0009 0003               NUMINP      EQU     $3
0010 0003               NUMOUT      EQU     $3
0011 8000               INPUT_MFS   EQU     $8000
0012 8060               SGLTN_POS   EQU     32*NUMINP+INPUT_MFS
0013 8078               RULE_START  EQU     8*NUMOUT+SGLTN_POS
0014                    *
0015                    * HC11 register block at $0000 in direct page
0016                    *
0017 000b               CFORC       EQU     $0B
0018 000c               OC1M        EQU     $0C
0019 000d               OC1D        EQU     $0D
0020 000e               TCNT        EQU     $0E
0021 0016               TOC1        EQU     $16
0022 0018               TOC2        EQU     $18
0023 001a               TOC3        EQU     $1A
0024 001c               TOC4        EQU     $1C
0025 0020               TCTL1       EQU     $20
0026 0022               TMSK1       EQU     $22
0027 0023               TFLG1       EQU     $23
0028 0024               TMSK2       EQU     $24
0029 0025               TFLG2       EQU     $25
0030 0026               PACTL       EQU     $26
0031 0030               ADCTL       EQU     $30             a/d control reg
0032 0031               ALPH_AD     EQU     $31             alpha a/d reg
0033 0032               BETA_AD     EQU     $32             beta a/d reg
0034 0033               GAMM_AD     EQU     $33             gamma a/d reg
0035 0034               DIST_AD     EQU     $34             distance a/d reg
0036 0039               OPTION      EQU     $39
0037 103d               INIT        EQU     $103D           note init reg is here after reset
0038 0000                           ORG     $0000       ;Beginning of HC11 RAM
0039 0000               REG_BLK     RMB     64
0040                    *
0041                    * Fuzzy engine RAM variables
0042                    *
0043 0040               CURRENT_INS RMB     NUMINP      ;Storage for 8-bit inputs
0044 0043               FUZ_INS     RMB     8*NUMINP    ;Storage for fuzzy inputs (RIGHT NOW MAX=8)
0045 005b               FUZ_OUTS    RMB     8*NUMOUT    ;Storage for fuzzy outputs (RIGHT NOW MAX=4)
0046 0073               COG_OUTS    RMB     NUMOUT      ;Defuzzified outputs
0047 0076               SUM_OF_FUZ  RMB     2           ;11-bit sum of fuzzy outs
0048 0078               SUM_OF_PROD RMB     3           ;19-bit sum of products
0049 007b               COGDEX      RMB     1           ;Current out # for COG loop
0050 007c               LP_COUNT    RMB     1           ;Index for fuz loop
0051 007d               SUMDEX      RMB     1           ;Index for sum loop
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0052                        *
0053                        *  User RAM variables
0054                        *
0055 007e                   BETA        RMB     2
0056 0080                   STEER       RMB     2
0057 0082                   SPEED       RMB     2
0058 0084                   DIRN        RMB     1
0059                        *
0060                        * Address Equates
0061                        *
0062 01ff                   STACK       EQU     $1FF
0063 00df                   OC1VEC      EQU     $DF
0064                        *
0065                        *  Buffalo routines
0066                        *
0067 ffd0                   .VECINIT    EQU     $FFD0
0068                        *
0069                        * Parameters
0070                        *
0071 6fff                   OC1_PER     EQU     $6FFF           period of OC1 timer - 14 ms works well
0072 0d80                   STR_MID     EQU     $0D80           mid pos of steer servo
0073 00f8                   LIMIT       EQU     $F8             end of tether
0074                        *
0075                        *  Give names to fuzzy engine i/p's and o/p's
0076                        *
0077 0040                   AB_FUZ      EQU     CURRENT_INS     a-b i/p to fuzzy engine
0078 0041                   GAM_FUZ     EQU     CURRENT_INS+1   gamma i/p to fuzzy engine
0079 0042                   DIST_FUZ    EQU     CURRENT_INS+2   distance i/p to fuzzy engine
0080 0073                   STR_FUZ     EQU     COG_OUTS        steer o/p from fuzzy engine
0081 0074                   SPD_FUZ     EQU     COG_OUTS+1      speed o/p from fuzzy engine
0082 0075                   DIR_FUZ     EQU     COG_OUTS+2      direction o/p from fuzzy engine
0083                        *
0084                        * User code starts here.....
0085                        *
0086 b600                               ORG     $B600           beginning of HC11 EEPROM
0087 b600 8e 01 ff          IO_INIT     LDS     #STACK           load stack
0088 b603 7f 10 3d                      CLR     INIT            move reg block to $0000
0089                        *
0090                        *  setup A/D
0091                        *
0092 b606 14 39 93                      BSET    OPTION $93      enable A/D
0093 b609 14 30 34                      BSET    ADCTL $34       setup A/D system
0094                        *
0095                        *  setup OC1 and OC2
0096                        *
0097 b60c 86 7e                         LDAA    #$7E
0098 b60e 97 df                         STAA    OC1VEC
0099 b610 cc b6 8a                      LDD     #OC1SVC
0100 b613 dd e0                         STD     OC1VEC+1        setup OC1 vector
0101 b615 14 0c 70                      BSET    OC1M $70
0102 b618 14 0d 70                      BSET    OC1D $70        OC1 will turn OC2/3/4 on
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0103 b61b 14 20 80              BSET    TCTL1 $80       setup OC2 to turn off
0104 b61e dc 0e                 LDD     TCNT
0105 b620 dd 16                 STD     TOC1            wait a full cycle
0106 b622 14 22 80              BSET    TMSK1 $80       enable OC1 int
0107                    *
0108                    * call Buffalo to setup pseudo vecs
0109                    *
0110 b625 bd ff d0              JSR     .VECINIT        initialize vectors (Buffalo)
0111                    *
0112                    *   init user vars
0113                    *
0114 b628 4f                    CLRA
0115 b629 97 7e                 STAA    BETA            clear Beta pre-byte
0116 b62b 97 82                 STAA    SPEED           set speed off
0117 b62d 43                    COMA
0118 b62e 97 84                 STAA    DIRN            set dirn reverse initially
0119 b630 cc 0d 80              LDD     #STR_MID
0120 b633 dd 80                 STD     STEER           set steer at middle
0121 b635 0e                    CLI                     unmask interrupts
0122                    *
0123                    * Main routine loops here
0124                    *
0125                    *   Normalize I/p parameters
0126                    *
0127 b636 4f            IP_NORM  CLRA
0128 b637 d6 32                 LDAB    BETA_AD
0129 b639 d7 7f                 STAB    BETA+1
0130 b63b d6 31                 LDAB    ALPH_AD
0131 b63d 93 7e                 SUBD    BETA            subtract beta from alpha
0132 b63f c3 00 7f      SCALE    ADDD    #$7F            add 127
0133 b642 2a 03                 BPL     BIG_AB
0134 b644 5f                    CLRB                    underflow
0135 b645 20 05                 BRA     SAVE_AB
0136 b647 4d            BIG_AB   TSTA
0137 b648 27 02                 BEQ     SAVE_AB
0138 b64a c6 ff                 LDAB    #$FF            overflow
0139 b64c d7 40         SAVE_AB  STAB    AB_FUZ
0140 b64e 96 33                 LDAA    GAMM_AD
0141 b650 97 41                 STAA    GAM_FUZ
0142 b652 96 34                 LDAA    DIST_AD
0143 b654 97 42                 STAA    DIST_FUZ
0144 b656 12 84 80 03           BRSET   DIRN $80 GO_FUZ check for forward
0145 b65a 73 00 41              COM     GAM_FUZ         trick to use REV fuzzy rules in FWD mode
0146                    *
0147                    * Call Fuzzy Engine
0148                    *
0149 b65d 8d 54         GO_FUZ   BSR     FUZZIFY
0150                    *
0151                    * Check distance limits
0152                    *
0153 b65f 96 42                 LDAA    DIST_FUZ
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0154 b661 27 04                 BEQ     STOP
0155 b663 81 f8                 CMPA    #LIMIT
0156 b665 25 0b                 BLO     ADJUST_OP
0157                    *
0158                    * Shutdown if at dock or at end of tether
0159                    *
0160 b667 14 0b 80      STOP    BSET    CFORC $80       stop engines
0161 b66a 4f                    CLRA
0162 b66b 97 0c                 STAA    OC1M
0163 b66d 97 22                 STAA    TMSK1
0164 b66f 97 20                 STAA    TCTL1
0165 b671 3f                    SWI                     does stop thru Buffalo
0166                    *
0167                    *  Adjust fuzzy outputs
0168                    *
0169                    *  Steering
0170                    *
0171 b672 4f            ADJUST_OP  CLRA
0172 b673 d6 73                 LDAB    STR_FUZ
0173 b675 05                    LSLD
0174 b676 05                    LSLD                    range of $07XX
0175 b677 05                    LSLD
0176 b678 c3 09 80              ADDD    #$980           max of $10XX
0177 b67b dd 80                 STD     STEER
0178                    *
0179                    *  Speed
0180                    *
0181 b67d 5f                    CLRB
0182 b67e 96 74                 LDAA    SPD_FUZ
0183 b680 dd 82                 STD     SPEED
0184                    *
0185                    *  Direction
0186                    *
0187 b682 96 75                 LDAA    DIR_FUZ
0188 b684 27 b0                 BEQ     IP_NORM         no change if zero
0189 b686 97 84                 STAA    DIRN
0190 b688 20 ac                 BRA     IP_NORM         loop to begin
0191                    *
0192                    * OC1 interrupt service routine generates pulse for
0193                    * steering servo and main engine
0194                    *
0195 b68a 15 23 7f      OC1SVC     BCLR    TFLG1 $7F    clear int flag
0196 b68d dc 16                 LDD     TOC1            get last compare
0197 b68f d3 80                 ADDD    STEER
0198 b691 dd 18                 STD     TOC2            update steer pulse width
0199 b693 dc 16                 LDD     TOC1
0200 b695 c3 6f ff              ADDD    #OC1_PER
0201 b698 dd 16                 STD     TOC1            update OC1 cycle
0202 b69a 93 82                 SUBD    SPEED
0203 b69c 7d 00 84              TST     DIRN
0204 b69f 2a 09                 BPL     FWD
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0205 b6a1 dd 1c          REV       STD     TOC4            update engine pulse
0206 b6a3 15 20 20                 BCLR    TCTL1 $20       de-program OC3
0207 b6a6 14 20 08                 BSET    TCTL1 $08       program OC4
0208 b6a9 3b                       RTI
0209 b6aa dd 1a          FWD       STD     TOC3            update engine pulse
0210 b6ac 15 20 08                 BCLR    TCTL1 $08       de-program OC4
0211 b6af 14 20 20                 BSET    TCTL1 $20       program OC3
0212 b6b2 3b                       RTI
0213                     ***** Fuzzy Inferrence Engine Starts Here
0214                     * Program assumes at least 1 valid input
0215                     *
0216 b6b3 18 ce 00 43    FUZZIFY   LDY     #FUZ_INS                ;Point to fuzzy input table
0217 b6b7 ce 00 42                 LDX     #CURRENT_INS+NUMINP-1   ;Point to last input value
0218 b6ba a6 00          PUSH_LOOP LDAA    0,X                     ;get value
0219 b6bc 36                       PSHA                            ;and store on stack
0220 b6bd 8c 00 40                 CPX     #CURRENT_INS            ;last value?
0221 b6c0 27 03                    BEQ     DONE_PUSH               ;if so, then branch
0222 b6c2 09                       DEX                             ;else point to next byte
0223 b6c3 20 f5                    BRA     PUSH_LOOP               ;branch for next value
0224
0225 b6c5 ce 80 00       DONE_PUSH LDX     #INPUT_MFS    ;Point to input MF specs
0226 b6c8 32             NXTIN_LP  PULA                  ;Get next current input value
0227 b6c9 c6 07                    LDAB    #7            ;For 8 loop passes
0228 b6cb d7 7c                    STAB    LP_COUNT      ;initialize loop counter
0229 b6cd                GRAD_LOOP EQU     *             ;Get grade for 1 label of 1 input
0230
0231                     ********************************************************************
0232                     * GET_GRADE - Routine to project a discrete input value onto       *
0233                     *   the associated input membership function (fuzzification).      *
0234                     *   Enter with input value in A register and X pointing at the     *
0235                     *   points and slopes defining the membership function.            *
0236                     *   Finishes with grade in B, X points at next MF spec (X+4)       *
0237                     *   and A is unchanged.                                            *
0238                     ********************************************************************
0239 b6cd 36             GET_GRADE PSHA                    ;Save input value of A
0240 b6ce 5f                      CLRB                    ;In case grade = 0
0241 b6cf a0 02                   SUBA    2,X             ;Input value D pt2 -> A
0242 b6d1 23 10                   BLS     NOT_SEG2        ;If input < pt2
0243 b6d3 e6 03                   LDAB    3,X             ;Slope 2
0244 b6d5 27 1c                   BEQ     HAV_GRAD        ;Skip if vert slope
0245 b6d7 3d                      MUL                     ;(In D pt1) * slp2 -> A:B
0246 b6d8 4d                      TSTA                    ;Check for > $FF
0247 b6d9 27 03                   BEQ     NO_FIX          ;If upper 8 = 0
0248 b6db 5f                      CLRB                    ;Limit grade to 0
0249 b6dc 20 15                   BRA     HAV_GRAD        ;In limit region of seg 2
0250 b6de c0 ff          NO_FIX    SUBB    #$FF           ;B D $FF
0251 b6e0 50                      NEGB                    ;$FF D B
0252 b6e1 20 10                   BRA     HAV_GRAD        ;($FF D((In D pt2) * slp2))
0253 b6e3 ab 02          NOT_SEG2  ADDA    2,X            ;Restore input value
0254 b6e5 a0 00                   SUBA    0,X             ;Input value D pt1 -> A
0255 b6e7 25 0a                   BLO     HAV_GRAD        ;In < pt1 so grade = 0
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0256 b6e9 e6 01                 LDAB    1,X             ;Slope 1
0257 b6eb 27 04                 BEQ     VERT_SLP        ;Skip if vert slope
0258 b6ed 3d                    MUL                     ;(in D pt1) * slp1 -> A:B
0259 b6ee 4d                    TSTA                    ;Check for > $FF
0260 b6ef 27 02                 BEQ     HAV_GRAD        ;Result OK in B
0261 b6f1 c6 ff         VERT_SLP  LDAB   #$FF           ;Limit region or vert slope
0262 b6f3 08            HAV_GRAD  INX                   ;Point at next MF spec
0263 b6f4 08                    INX
0264 b6f5 08                    INX
0265 b6f6 08                    INX
0266 b6f7 32                    PULA                                    ;Restore A register
0267
0268 b6f8 18 e7 00              STAB    0,Y                             ;Save @ fuzzy input
0269 b6fb 18 08                 INY                                     ;Point at next
0270 b6fd 7a 00 7c              DEC     LP_COUNT                        ;
0271 b700 2a cb                 BPL     GRAD_LOOP                       ;For 8 labels of 1 input
0272 b702 18 8c 00 5b           CMPY    #NUMINP*8+FUZ_INS               ;Done with all fuzzy ins?
0273 b706 26 c0                 BNE     NXTIN_LP                        ;If not, process next input
0274                    * Done with fuzzification, evaluate rules next
0275
0276 b708 ce 00 5b              LDX     #FUZ_OUTS       ;Point at first fuzzy output
0277 b70b 86 18                 LDAA    #8*NUMOUT       ;Number of fuzzy outputs
0278 b70d 6f 00         CLR_OUTS  CLR    0,X            ;Clear a fuzzy output
0279 b70f 08                    INX                     ;Point at next
0280 b710 4a                    DECA                    ;Loop index
0281 b711 26 fa                 BNE     CLR_OUTS        ;Continue till all fuzzy outs 0
0282 b713 18 ce 80 78           LDY     #RULE_START     ;Point to start of 1st rule
0283 b717 86 ff         RULE_TOP  LDAA   #$FF           ;Begin processing a rule string
0284                    * A will hold grade of smallest (min) if part
0285 b719 18 e6 00      IF_LOOP   LDAB   0,Y            ;Get rule byte OOOX XAAA; if X is A
0286 b71c 2b 24                 BMI     THEN_LOOP       ;If MSB=1, exit to then loop
0287 b71e 18 08                 INY                     ;Point at next rule byte
0288 b720 ce 00 43              LDX     #FUZ_INS        ;Point at fuzzy inputs
0289 b723 3a                    ABX                     ;Point to specific fuzzy input
0290 b724 a1 00                 CMPA    0,X             ;Is this fuzzy input lower?
0291 b726 23 f1                 BLS     IF_LOOP         ;If not don't replace lowest
0292 b728 a6 00                 LDAA    0,X             ;Replace lowest if
0293 b72a 26 ed                 BNE     IF_LOOP         ;Unless zero, do next rule byte
0294                    * A zero value fuzzy input makes rule completely not true
0295                    * so skip rest of if parts and all then parts to start of next rule
0296 b72c 18 e6 00      FIND_THEN LDAB   0,Y            ;Get next rule byte
0297 b72f 2b 04                 BMI     FIND_IF         ;MSB set means its a then part
0298 b731 18 08                 INY                     ;Point at next rule byte
0299 b733 20 f7                 BRA     FIND_THEN       ;Loop till pointing at a then part
0300 b735 18 08         FIND_IF   INY                   ;Adv rule pointer to if part
0301 b737 18 e6 00              LDAB    0,Y             ;Get next rule byte
0302 b73a 2a db                 BPL     RULE_TOP        ;MSB clear means its an if part
0303 b73c c1 ff                 CMPB    #$FF            ;$FF is no more rules marker
0304 b73e 26 f5                 BNE     FIND_IF         ;Continue looking for if or $FF
0305 b740 20 17                 BRA     DEFUZ           ;When all rules done, go defuzzify
0306
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0307 b742 ce 00 5b      THEN_LOOP   LDX    #FUZ_OUTS     ;Point at fuzzy outputs
0308 b745 c4 7f                     ANDB   #$7F          ;Save 8 times out # + label #
0309 b747 3a                        ABX                  ;X points at fuzzy output
0310                    * Grade of membership for rule is in A accumulator
0311 b748 a1 00                     CMPA   0,X           ;Compare to fuzzy output
0312 b74a 25 02                     BLO    NOT_HIER      ;Branch if not higher
0313 b74c a7 00                     STAA   0,X           ;Grade is higher so update
0314 b74e 18 08         NOT_HIER    INY                  ;Adv rule pointer to next byte
0315 b750 18 e6 00                  LDAB   0,Y           ;Get rule byte
0316 b753 2a c2                     BPL    RULE_TOP      ;If MSB=0 its a new rule
0317 b755 c1 ff         CHK_END     CMPB   #$FF          ;Check for end of rules flag
0318 b757 26 e9                     BNE    THEN_LOOP     ;If not $FF, must be a then part
0319
0320                    * All rules evaluated. Now defuzzify fuzzy outputs
0321 b759 18 ce 80 60   DEFUZ       LDY    #SGLTN_POS    ;Point at 1st output singleton
0322 b75d ce 00 5b                  LDX    #FUZ_OUTS     ;Point at 1st fuzzy output
0323 b760 7f 00 7b                  CLR    COGDEX        ;Loop index will run from 0->1
0324 b763 c6 08         COG_LOOP    LDAB   #8            ;8 fuzzy outs per COG output
0325 b765 d7 7d                     STAB   SUMDEX        ;Inner loop runs 8->0
0326 b767 cc 00 00                  LDD    #$0000        ;Used for quicker clears
0327 b76a dd 76                     STD    SUM_OF_FUZ    ;Sum of fuzzy outputs
0328 b76c dd 79                     STD    SUM_OF_PROD+1 ;Low 16-bits of sum of products
0329 b76e 97 78                     STAA   SUM_OF_PROD   ;Upper 8-bits
0330 b770 e6 00         SUM_LOOP    LDAB   0,X           ;Get a fuzzy output
0331 b772 4f                        CLRA                 ;Clear upper 8-bits
0332 b773 d3 76                     ADDD   SUM_OF_FUZ    ;Add to sum of fuzzy outputs
0333 b775 dd 76                     STD    SUM_OF_FUZ    ;Update RAM variable
0334 b777 a6 00                     LDAA   0,X           ;Get fuzzy output again
0335 b779 18 e6 00                  LDAB   0,Y           ;Get Output singleton position
0336 b77c 3d                        MUL                  ;Position times weight
0337 b77d d3 79                     ADDD   SUM_OF_PROD+1 ;Low 16-bits of sum of products
0338 b77f dd 79                     STD    SUM_OF_PROD+1 ;Update low 16-bits
0339 b781 96 78                     LDAA   SUM_OF_PROD   ;Upper 8-bits
0340 b783 89 00                     ADCA   #0            ;Add carry from 16-bit add
0341 b785 97 78                     STAA   SUM_OF_PROD   ;Upper 8-bits of 24-bit sum
0342 b787 18 08                     INY                  ;Point at next singleton pos.
0343 b789 08                        INX                  ;Point at next fuzzy output
0344 b78a 7a 00 7d                  DEC    SUMDEX        ;Inner loop index
0345 b78d 26 e1                     BNE    SUM_LOOP      ;For all labels this output
0346 b78f 3c                        PSHX                 ;Save index for now
0347 b790 4f                        CLRA                 ;In case divide by zero
0348 b791 de 76                     LDX    SUM_OF_FUZ    ;Denominator for divide
0349 b793 27 18                     BEQ    SAV_OUT       ;Branch if denominator is 0
0350 b795 7d 00 78                  TST    SUM_OF_PROD   ;See if more than 16-bit
0351 b798 26 07                     BNE    NUM_BIG       ;If not zero, # is > 16-bits
0352 b79a dc 79                     LDD    SUM_OF_PROD+1 ;Numerator for divide
0353 b79c 02                        IDIV                 ;Result in low 8-bits of X
0354 b79d 8f                        XGDX                 ;Result now in B
0355 b79e 17                        TBA                  ;Move result to A
0356 b79f 20 0c                     BRA    SAV_OUT       ;Go save output
0357 b7a1 dc 78         NUM_BIG     LDD    SUM_OF_PROD   ;Numerator upper 16 of 24-bit
```

APPENDIX C – FUZZY TRUCK ASSEMBLY CODE

```
0358 b7a3 7d 00 7a                    TST     SUM_OF_PROD+2 ;Check for rounding error
0359 b7a6 2a 03                       BPL     NO_ROUND      ;If MSB clear, don't round
0360 b7a8 c3 00 01                    ADDD    #1            ;Round numerator up 1
0361 b7ab 03              NO_ROUND     FDIV                  ;D/X -> X, use upper 8 of 16
0362 b7ac 8f                          XGDX                  ;Result now in A
0363 b7ad ce 00 73        SAV_OUT      LDX     #COG_OUTS     ;Point to 1st defuz output
0364 b7b0 d6 7b                       LDAB    COGDEX        ;Curent output number
0365 b7b2 3a                          ABX                   ;Point to correct output
0366 b7b3 a7 00                       STAA    0,X           ;Update defuzzified output
0367 b7b5 38                          PULX                  ;Recover Index
0368 b7b6 5c                          INCB                  ;Increment loop index
0369 b7b7 d7 7b                       STAB    COGDEX        ;Update
0370 b7b9 c1 03                       CMPB    #NUMOUT       ;Done with outputs?
0371 b7bb 26 a6                       BNE     COG_LOOP      ;If not, continue loop
0372                      *****
0373                      * Inference engine has completed one pass of all rules.
0374                      *****
0375 b7bd 39                          RTS                   ;Patch Inserted by GJE for use as subroutine
```