Development of a Fuzzy Logic Speed and Steering Control System For an Autonomous Vehicle

Scott Pawlikowski

University of Cincinnati
Master of Science Research Project

Department of Mechanical Engineering
January 10, 1999

Table of Contents

List of Figures

List of Tables

1. Abstract

Autonomous vehicles have many potential applications in the fields of automation, defense and exploration. The purpose of this paper is to describe the development of a fuzzy logic propulsion and steering control algorithm for an autonomous vehicle. Using an integrated vision system, the vehicle senses position relative to the angle of a line drawn on the ground, and processes that information through a fuzzy logic algorithm. The algorithm selects drive speeds for two independent brushless DC motors, which are connected to the front wheels of the vehicle. By selecting the speed of each motor, the vehicle will be able to autonomously propel and steer, following a given path. The system was analyzed, modeled, and tuned to develop a fuzzy logic controller that properly achieves the steering and speed control design requirements.

2. Background

Fuzzy logic is one of the newest paradigms in the control system community. Originally invented in 1964 in America, fuzzy logic more closely represents human thought than the fundamentals of classical control. Fuzzy logic allows a user to describe a system using natural words, and create a control mechanism based on "expert rules" or in other words, rules developed through experience. A fuzzy controller is developed without creating a mathematical model of the system. For example, a fuzzy logic code may contain the following linguistic rule:

If the angle of the input line is high, then turn the vehicle quickly to the right.

The blend of these simple rules allow the system to be easily analyzed, controlled, modified, and understood.[15]

Initially, the concepts of Fuzzy Logic were very controversial in the American Academic Community. However, in the thirty years since the development of Fuzzy Logic, the Japanese have embraced the powerful simplicity of the technology, and developed control systems for products as diverse as self-focusing cameras, dishwashers, and subway systems, and are now earning billions of dollars selling the technology back to the Americans. The power of fuzzy logic lies in its simplicity, and with its close resemblance with the human thought process.[9]

The foundations of Fuzzy Logic are as old as the discipline of the traditional logic theory Aristotle developed centuries ago. Classical logic theory is based on the principle of "crisp" sets, or the law of the excluded middle which says X must be in set A or in set not-A. Everything falls in one group or another; there is nothing that is both a day of the week and not a day of the week. Often, classical set theory assigns a membership value to the universe -- 1 for true statements, 0 for false.

Even Aristotle recognized the flaw in this reasoning. Human thought allows consideration of vagueness and imprecision. The concepts of fuzzy logic attempt to describe the universe in a more natural manner. Lofti Zadeh, a professor at the University of California at Berkley recognized these flaws with traditional logic theory, and invented the concepts of fuzzy logic in 1964.[16]

Fuzzy logic allows each element of the set being considered, or the **universe of discourse**, to belong to a set *to a certain degree*. This degree of belonging is represented in fuzzy sets by a membership value that varies between 0 and 1. Of course, classical set theory is a special case of fuzzy set theory that contains a discontinuous break in the membership of a set in the jump between true and not true. Fuzzy logic allows a more continuous, smooth variation from membership to non-membership in a set. **A membership function** represents how each point in the input space is mapped with a degree of membership to the set in consideration. Membership functions can come in many shapes from bell curves, to s-functions, to even crisp "classical" membership functions. Although shapes of membership functions vary, ranges of the function usually varies between 0 and 1.

In a vehicle control algorithm, one of the important aspects is the distance relative to some reference object. In the case of Bearcat Jr., the reference object is a set of boundary lines, 10 feet apart. Therefore, the universe of discourse of the distance measurement is the continuous set of distances between 0 and 10 feet.

Using the concepts of classical set theory, a statement could be made that asserts distances closer than two feet are considered near the boundary line. This assertion works very near the boundaries of the set; distances that are above 7 feet are considered not-near, and distances less than 1 feet are considered near, however the logic begins to break down near the transition point. It seems unreasonable to call a distance "near" that is just below 2 feet and a distance not near just above 2 feet, especially if these distances differ by a fraction of an inch. Fuzzy Logic allows the *membership function* of the set of near distances to vary continuously from 0 feet to 10 feet, from near to not near. Graphical descriptions of the "crisp" and "fuzzy" membership functions are shown below.[9]

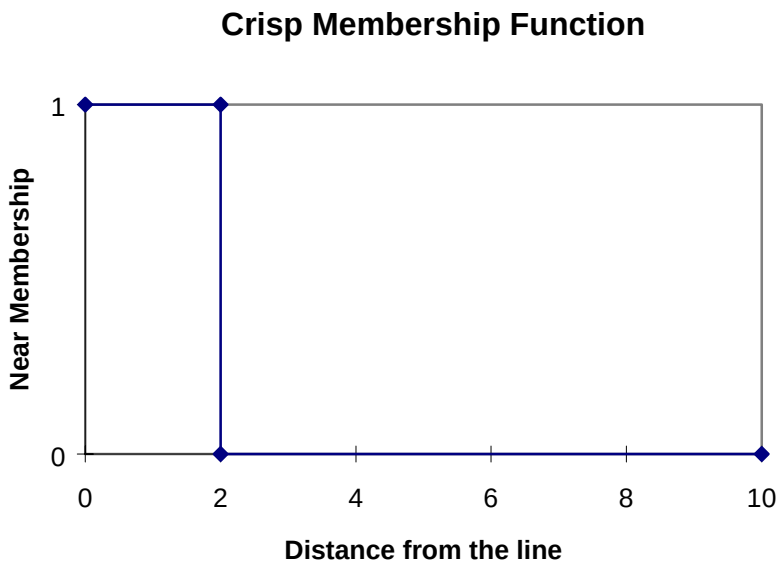


Figure 1: Example of a crisp membership function

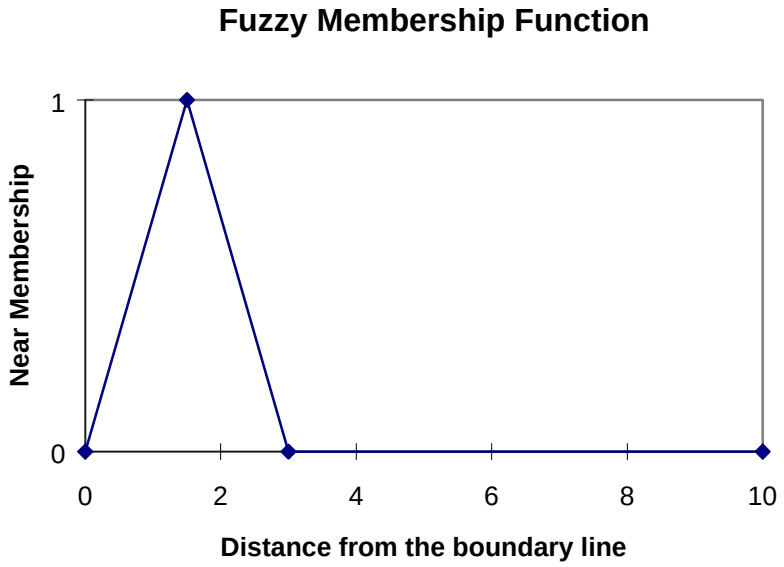


Figure 2: Example of a fuzzy membership function

The principles of classical set theory, and classical logic are translated into fuzzy set counterparts through the use of mathematical relations. Of particular importance are the principles set intersection, set union and conditional statements.

In classical set theory, the principle of set intersection is used to identify common membership between 2 sets. Intersection is usually indicated through an AND statement. To be a member of a set described by an AND statement, the item under consideration must be a member of both sets combined with the AND statement. For example, the following categories represent a distance considered to be near the boundary, a distance considered to be ideal, and a distance considered to be both near the boundary and ideal (1 indicates set membership, 0 indicates non-membership.)

Table 1: Example of classical logic AND function

Near Boundary	Ideal Distance	Near Boundary and Ideal Distance
0	0	0
0	1	0
1	0	0
1	1	1

In fuzzy set theory, the process of intersection is interpreted through the use of the mathematical minimum function, which returns the minimum of two values. The minimum function can also be used to interpret AND statements in the Classical Logic example. For example, the following categories represent a distance considered to be near the boundary, a distance considered to be ideal, and a distance considered to be both near the boundary and ideal.

Table 2: Example of fuzzy logic AND function

Near Boundary	Ideal Distance	Near Boundary and Ideal Distance
.2	.5	.2
.1	.05	.05
1.0	0	0
.1	.5	.1

In classical set theory, the principle of set union is used to identify the entire membership of 2 sets. Union is usually indicated through an OR statement. To be a member of a set described by an OR statement, the item under consideration must be a member of either of the sets combined with the OR statement. For example, the following categories represent a motor speed to be slow forward, forward, and slow forward or forward.

Table 3: Example of classical logic OR function

Slow Forward	Forward	Slow Forward or Forward
0	0	0
0	1	1
1	0	1
1	1	1

In fuzzy set theory, the process of intersection may be handled by the use of the mathematical maximum function, which returns the maximum of two values. For example, the following categories represent a motor speed to be slow forward, forward, and slow forward or forward.

Table 4: Example of fuzzy logic OR function

Slow Forward	Forward	Slow Forward or Forward
.2	.5	.5
.1	.05	.1
1.0	.4	1.0
.9	1.0	1.0

Again, note that the maximum function can be applied to the Classical Logic example.

In system control, decisional algorithms are often used to issue commands from ongoing feedback in the system. Decisional algorithms are comprised of conditional statements which are used to describe the state of an output when a system is subjected to a given input.[9] Conditional statements are usually made in the form of if - then statements. For example, one of the fuzzy logic rules for control of Bearcat Jr. is stated as follows:

> If the vehicle is right camera controlled, and the distance is near, and the angle is positive, then the right motor speed is slow forward, and the left motor speed is forward

The output of these statements occurs if and when the antecedent or input is true. In fuzzy logic, since inputs can be partially true, this evaluation becomes more complicated. Fuzzy logic theory suggests that if an input is partially true, then the output is partially true to the same degree. One of the most common ways to perform this operation is to truncate the output membership function. For example:

A simplistic rule could state that

If the distance from the boundary is near, than the right motor velocity is forward.

Membership functions are visually represented for both the output and the antecedent. The input is at a distance of 2.0 feet, which corresponds to a membership of .4 to the near membership function. The net result of this conditional statement is a new membership function centered at the center of the output "forward" membership function, with an area related to the degree of membership of the input function. The fuzzy output is translated to a crisp output by finding the centroid of this output membership area.

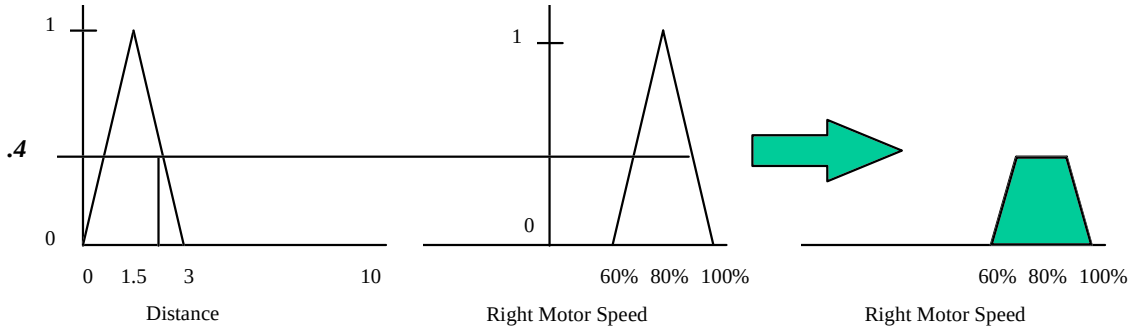


Figure 3: Example of fuzzy conditional logic

A fuzzy logic controller utilizes an algorithm of overlapping patches of these conditional statements to define desired outputs for each of the given inputs in the design space. These patches are often simpler to create and utilize than the elaborate mathematical models that are often created to describe a complex, non-linear system.

These three components form the building blocks of every fuzzy logic system: membership functions, combination functions (and / or), and conditional statements. These components are combined together to form a system that translates a crisp input to a crisp output. To allow multiple rules to effect the same output, an aggregate method is used to combine all of the calculated outputs into 1 crisp output value. The aggregate method allows for the blend of several fuzzy logic rules. Mathematically, each fuzzy output (represented by an area) is linearly combined. The centroid of the resulting area is computed, and the result is the crisp output of the controller. The following diagram represents all of the steps that form the foundation of the control output: Distance information is input into the controller, which fuzzifies the input, translates that input to a motor speed output based on conditional statements, and combines the outputs from several rules using an aggregate method to determine a crisp output.[13]

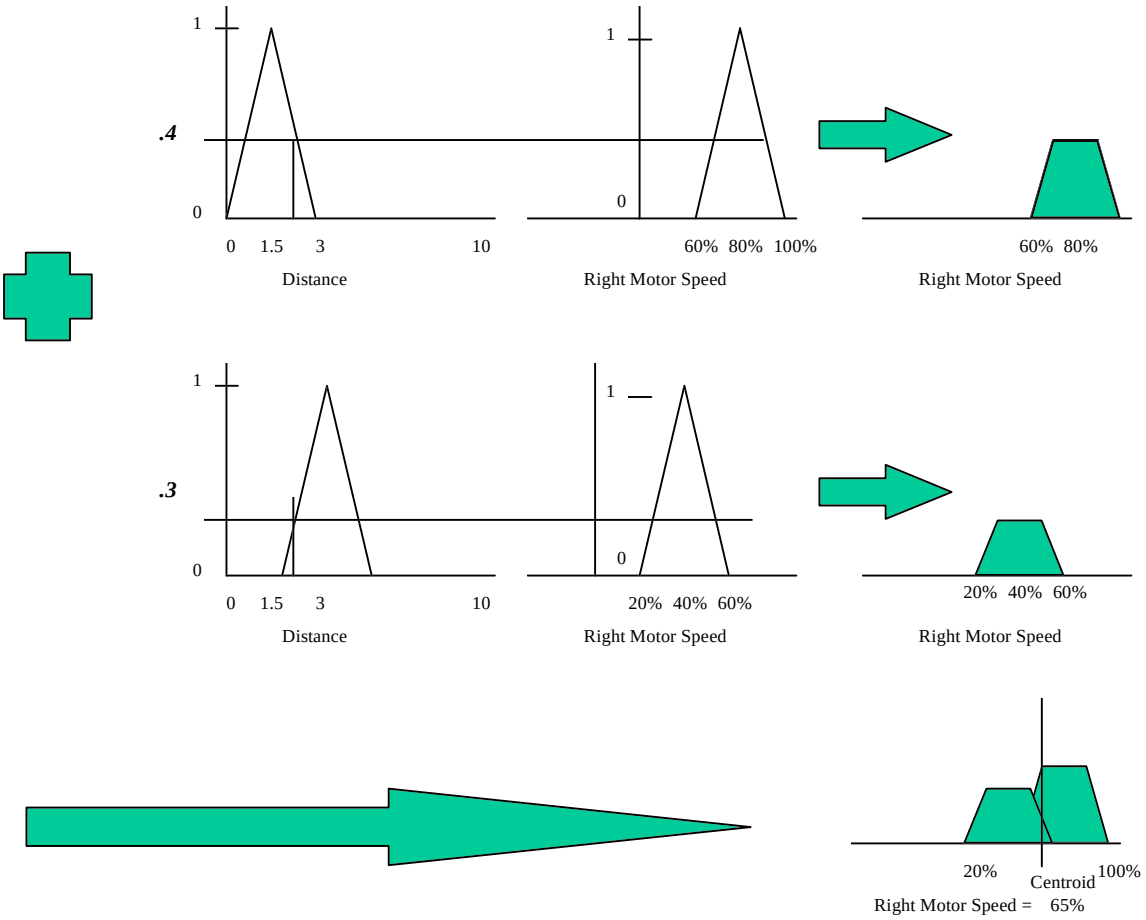


Figure 4: Steps of complete fuzzy logic system

As described in the Matlab Fuzzy Logic Toolbox Handbook, this process is summarized into the following 5 process steps:

1. **"Fuzzify" the inputs**: The first step is to evaluate to what degree does each input belong to each of the appropriate input fuzzy sets. Therefore, the first part of the development of a fuzzy logic controller must be to develop the membership functions to translate each "crisp" input, which can be any number depending on the input signal, to a "fuzzified" variable which is contained between 0 and 1.

2. **Apply the fuzzy operator:** If the antecedent of the fuzzy rules contains a combination, such as a union (OR) , or an intersection (AND) of several of the fuzzified inputs, the next step is to apply the fuzzy operator to translate the input statement into one input value. This may be done using the minimum function for intersection statements (AND), and the maximum function for union (OR) statements.

3. **Apply the implication method**: The implication method translates the value of the antecedent of a conditional statement to a value of the output. The minimum function is used to truncate the output membership function of the output fuzzy set.

4. **Aggregate all outputs:** Since several rules may describe a single output, a method is required to combine, or aggregate, all of the rules into a single fuzzy value. Each of the truncated outputs from the implication method are combined together, using a maximum or even a sum of the set of truncated output membership functions.

5. **Defuzzify:** The final step is to translate the aggregate of all of the fuzzy outputs into a single, crisp, real output value. The most popular defuzzification method is the centroid method, which returns the center of the area under the curves representing all of the truncated membership functions.[4]

These process steps have been implemented to create a controller that allows the autonomous propulsion of the Bearcat Jr. vehicle.

3. Description of the Autonomous Vehicle

The system that is to be controlled is an electrically propelled mobile vehicle nicknamed Bearcat Jr. Bearcat Jr. was created and assembled during the Spring quarter of 1998 in the Advanced Robotics Lab at the University of Cincinnati. This vehicle was built as part of the Autonomously Guided Vehicle contest sponsored by the Army Tank Command. A 3D rendering of Bearcat Jr. is shown below. The vehicle is constructed of an aluminum frame designed to hold the controller, obstacle avoidance, vision sensing, vehicle power system, and drive components. Two independently driven brushless DC motors are used for both vehicle propulsion, as well as for vehicle steering.

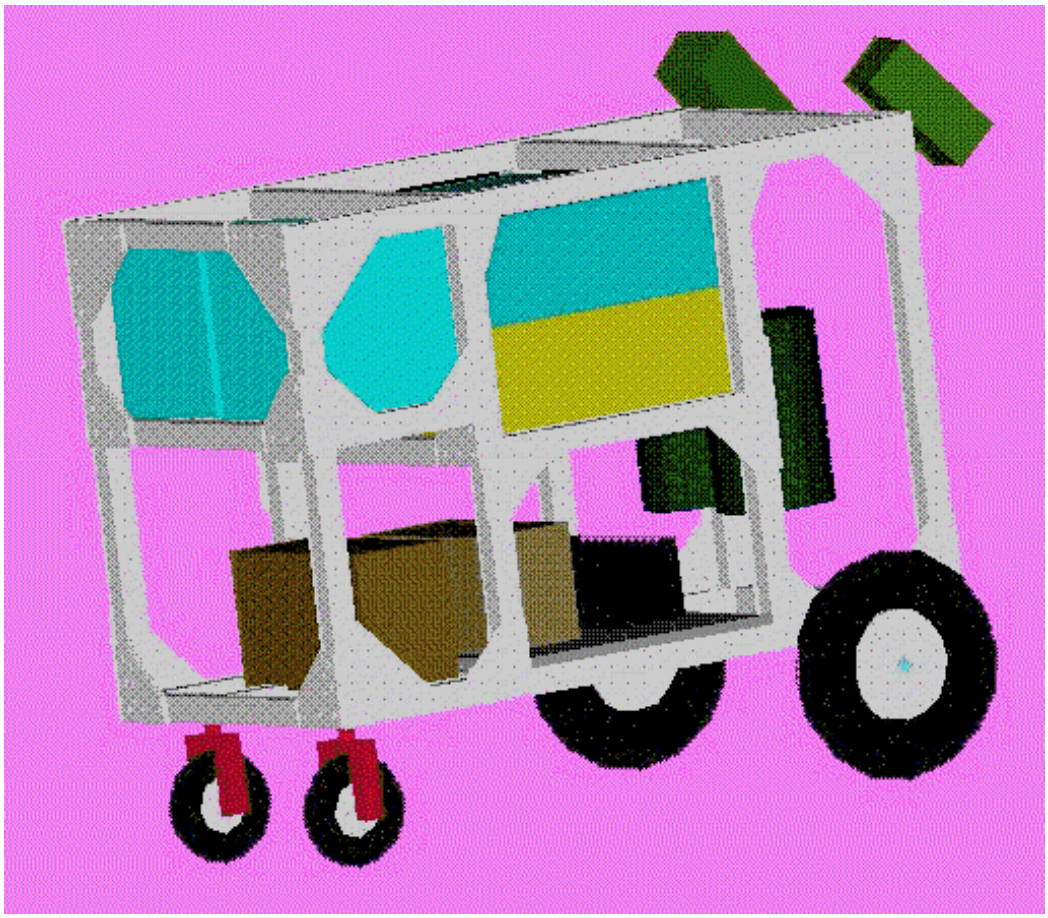


Figure 5: The Bearcat Jr. Vehicle

This independent drive system not only gives the Bearcat Jr. vehicle the capability to move in a straight line, and perform turns, but this system also allows the vehicle to have a zero turning radius feature. This feature allows the vehicle to turn directly about the center of the drive without requiring forward motion, thereby giving the vehicle the ability to navigate through more complicated course requirements.

A 3-axis Gallil motion control board is used as the interface between the controller CPU and the drive

components, including the brushless servo drive motors and the encoders. A Galill board, the brushless motor, and an optical encoder provide a closed loop system that allows the controller code to specify accurately motor dynamics parameters, including position, velocity and acceleration. The Gallil controller contains a digital PID type controller. This controller is tuned with the Servo Design Kit Software package, that selects the Proportional, Integral, and Derivative gains (Kp, Ki, Kd) to optimize the system response. Once tuned, the controller code is able to simply select motor position, velocity and acceleration, and control the trajectory of the Bearcat Jr. vehicle. It is important to distinguish these PID components from the control logic used for vehicle: this controller drives the motor to a specified velocity, whereas the control logic selects the value of that specified velocity based on vision system inputs.

There are two control system types that may be used to control an autonomously guided vehicle. The first is to provide the vehicle with complete information about the environment, and the required path. The vehicle then uses navigation sensors, and the programmed information about the environment to navigate through the course. This method requires extensive programming to completely define the path, and is unable to navigate in an unknown territory. The second method is to gather information from the environment using external sensors, and process the information to control the speed and steering parameters of the vehicle. Due to the unknown environment for the usage of the Bearcat Jr. vehicle, this second method is utilized.

For path generation, the input to the controller acquired through 2 cameras mounted on the front of the vehicle. These cameras sense the visual image of the path-line. This information is processed through the ISCAN system. The ISCAN system returns two variables:

- the distance of the vehicle from the border line
- and the relative angle between the path of the vehicle and the border line.

The controller utilizes these two inputs to select motor speeds that will drive the vehicle to follow the specified path without crossing the boundary lines. The vehicle can be driven in a straight line by specifying equivalent angular velocities of the motors, or driven in a turn by specifying differing angular velocities of the motors.

A schematic of the components of the control system is shown below:

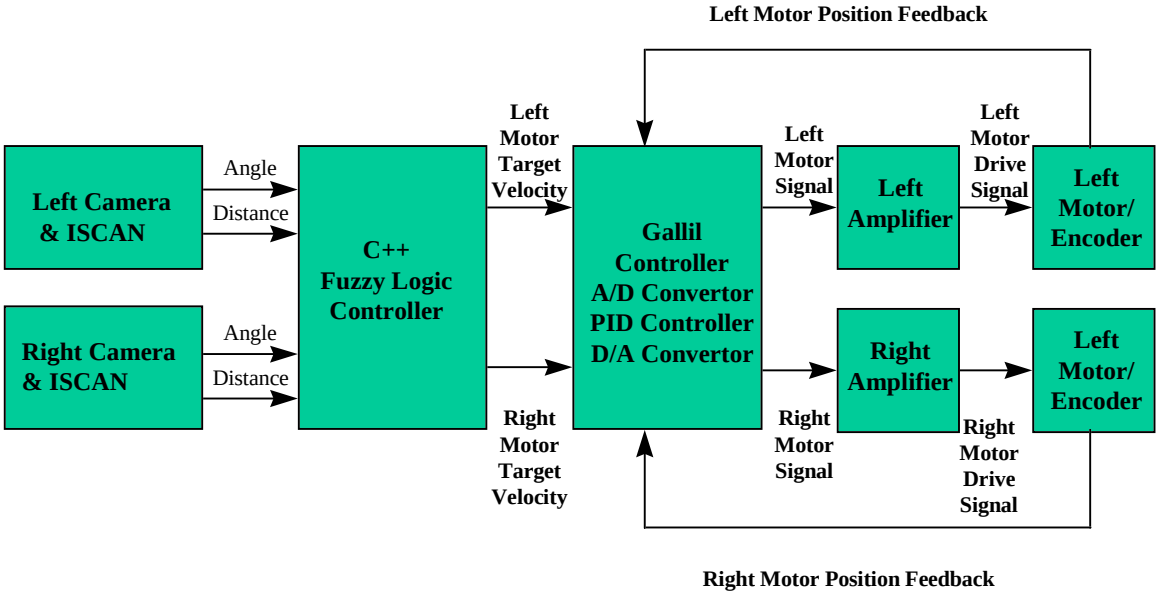


Figure 6: Control System Schematic

Through a fuzzy logic algorithm, the code translates the input from the vision system into target output velocities for each of the right and left vehicle drive motors. These outputs are processed through the Gallil motion control system that translates the target output into controller output signals. These signals are passed through the amplifiers to increase the signals to a level that will drive the motors at the proper velocity. The encoders, mounted on the ends of the motors, supply an angular position feedback signal back to the Galill control board to minimize

the steady state error between the target velocity, and the actual motor velocity. The end result is the desired vehicle motion at the target velocity. This system provides motor output signals for each input sample. The system is therefore able to adjust to a changing environment, and changing path conditions.[1]

The previous control logic for the Bearcat Jr. vehicle utilized a classical linear control system with feedback, and a PID (Proportional, Integrator, Derivative) controller. This system was selected due to the simplicity of its components, and the breadth of knowledge of the team concerning the design and analysis of these systems. An error signal is measured, between the desired and actual state of the system. Through the linear control system, this error signal is driven to a zero value, and the desired state is achieved. This system provided several challenges for the team, focusing mainly on the complexity of the multiple input / multiple output, and the difficulty of tuning the system in response to changing environments. For this reason, a fuzzy logic controller, which is simpler to understand, and to modify, was created.

To control the vehicle, a control algorithm is developed that can easily be coded into C++, which is the language for the Gallil control hardware interface. The Gallil hardware is supplemented with C++ software that allows direct specification of motor velocity by calling the appropriate “speed” subroutine. The Gallil board translates these speed commands into output signals which, after proper tuning, result in a desired motor angular position, velocity and acceleration. [2]

4. Description of Control Logic

The first step is developing the membership functions for the fuzzification and defuzzification steps of the fuzzy logic process. The following membership functions are used to translate real conditions into fuzzy logic values:

1.) Distance: The distance membership function is a three-pronged triangular membership function representing vehicle conditions that are good, near and far from the boundary lines. Since the path is 10 feet wide and the vehicle is 4 feet wide, the edge of the vehicle should ideally be 3 feet from the boundary line to be in the center. Any value within a foot of this position is considered to be good. The near and far membership functions overlap the good membership functions, so that the vehicle can be both good, and near the line, or both good and far from the line. This overlap in the membership functions allows application of multiple rules in this region. This membership function is used for both right camera and left camera inputs.

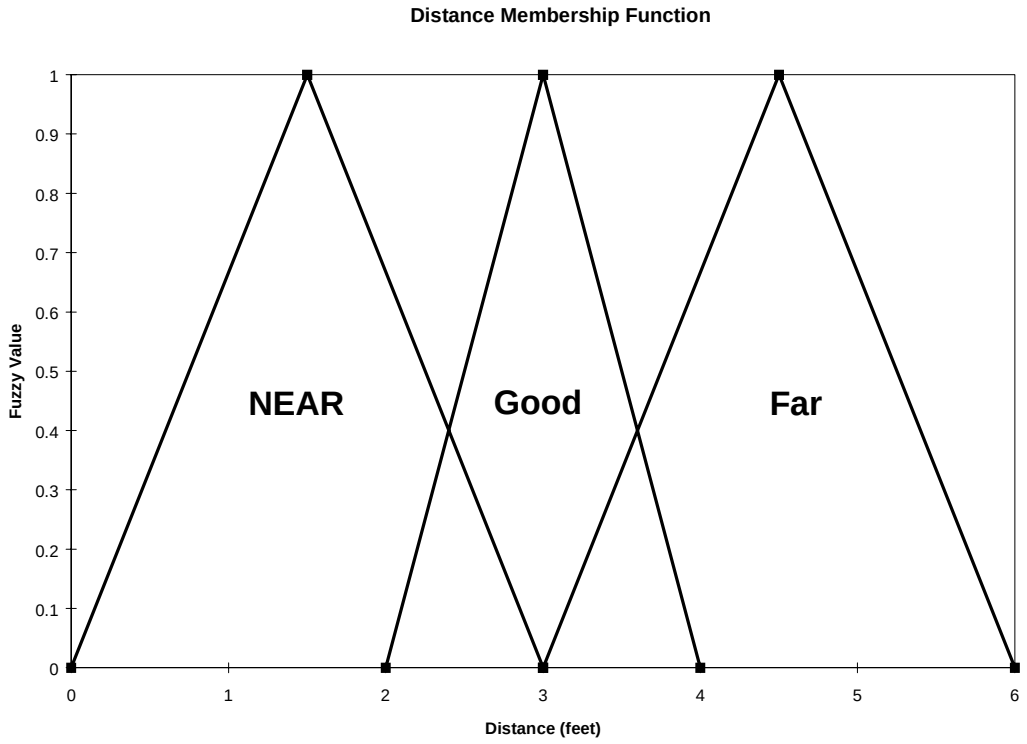


Figure 7: Distance membership functions

2.) Angle: The angle membership function is a 5-pronged triangular membership function, representing angle conditions that are very negative, negative, zero, positive, and very positive. The ISCAN system only has a range of 180 degrees, therefore the angular membership functions are defined between -90 and 90 degrees. Within the region of -20 to 20 degrees, an angle will have membership to both the zero membership function and either of the negative or positive membership function.

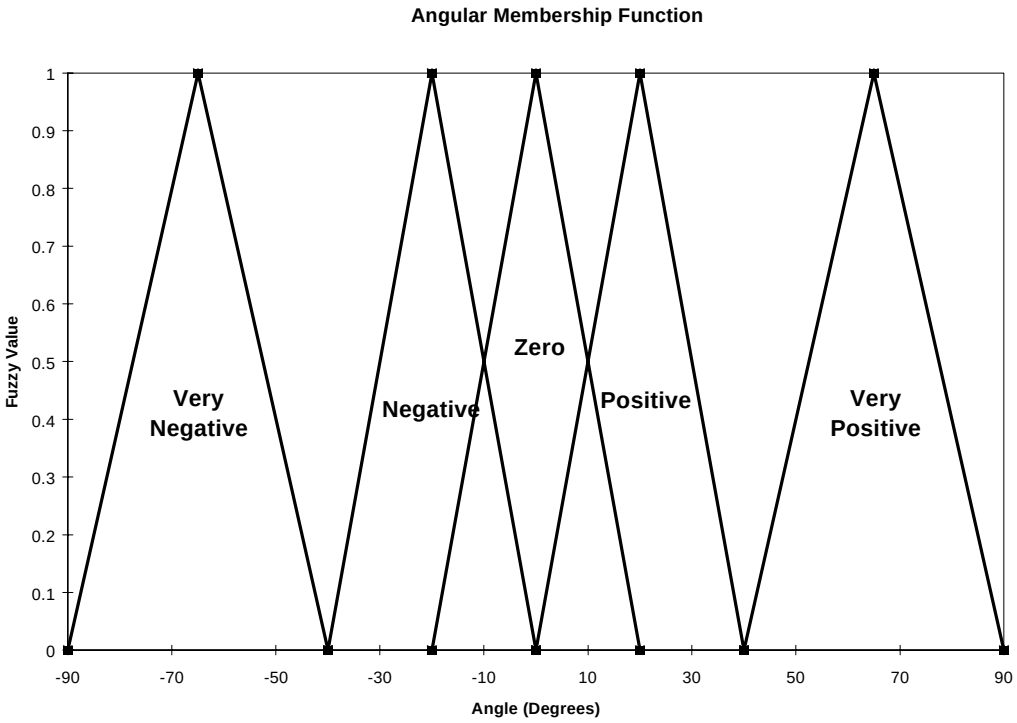


Figure 8: Angle membership functions

3.) Motor Speed: The motor speed membership function is a 5-pronged triangular membership function, representing motor speed conditions that are backward, slow backward, stop, slow forward and forward,. negative, zero, positive, and very positive. The motor membership functions are defined as a percentage of the maximum motor speed from -100% to 100%. These membership functions contain a factor of safety for the motor speed. Since the centroid of the forward and backward membership functions is at 80%, the motor speed will never operate at more than 80% of the maximum rated speed. Since motor speed is an output of the system, these membership functions do not overlap. The left and right motors have identical membership functions.

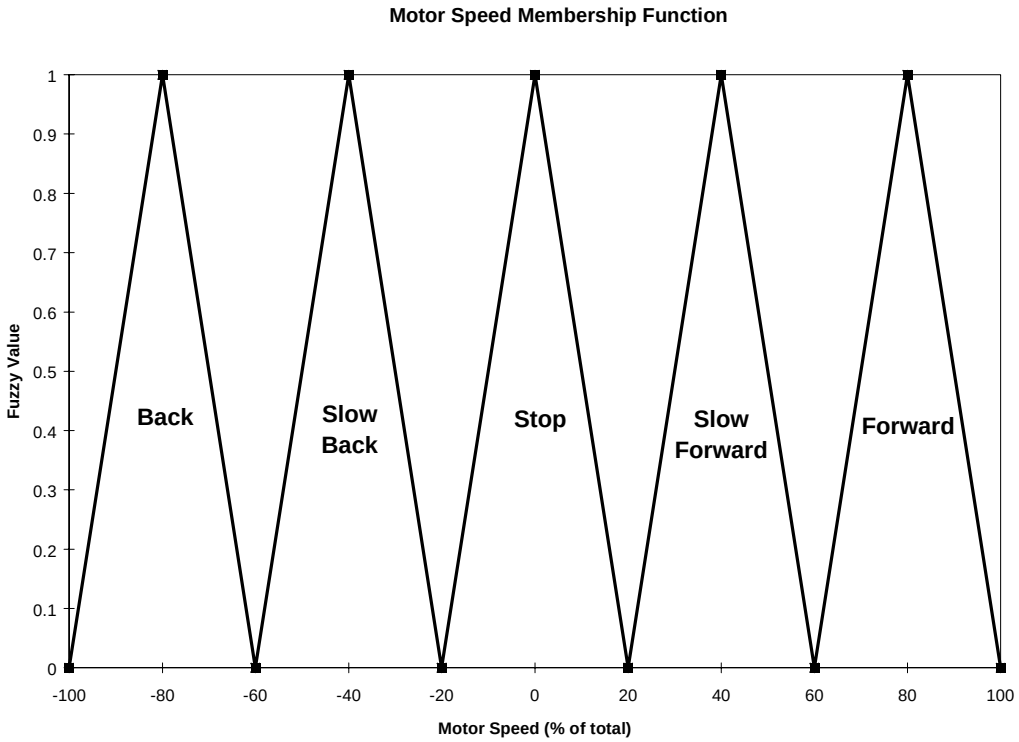


Figure 9: Motor speed membership functions

The next step is developing appropriate fuzzy logic rules, so the performance requirements of the vehicle must be defined. The overall goal is to propel the vehicle through the course, following the path defined by the white boundary lines. Based on the experience of the team, and this performance objective, the following control logic was developed to select the velocity of each motor, and in turn, the velocity and trajectory of the vehicle to successfully navigate the course.

The Bearcat Jr. Vehicle has 2 cameras, left and right, to sense the position of the vehicle relative to the path boundary lines.[4] The first section of the speed and steering controller code selects which camera data will be used to sense the input of the cart. Since the primary goal of the course is to navigate the path, and avoid crossing the boundary lines, the nearest boundary line is the critical reference input. Therefore, the controller code first compares the relative distance of the two lines, and selects the camera submitting the lowest distance data, and therefore nearest boundary line, as the controller input. If, in the exceptional case, one camera does not submit any information, or the two cameras submit identical data, the primary input will remain the same as the prior iteration. This camera selection process is the first step of each of the speed selection loop iterations. The input camera is the first of the inputs to the control code. The controlling camera directs the control code into one of two mirror image control logic schemes, one representing right camera control, the other representing left camera control.

Right Camera Control

The camera, and ISCAN system submit only angle and distance information about the position of the vehicle. The underlying principle of all of the controller logic is to bring the Bearcat Jr. vehicle into the center of the track, parallel to the path boundary line, as measured by the vision system. By programming a fuzzy logic system into the controller code, this information is used to select the motor speed selection patterns, as follows.

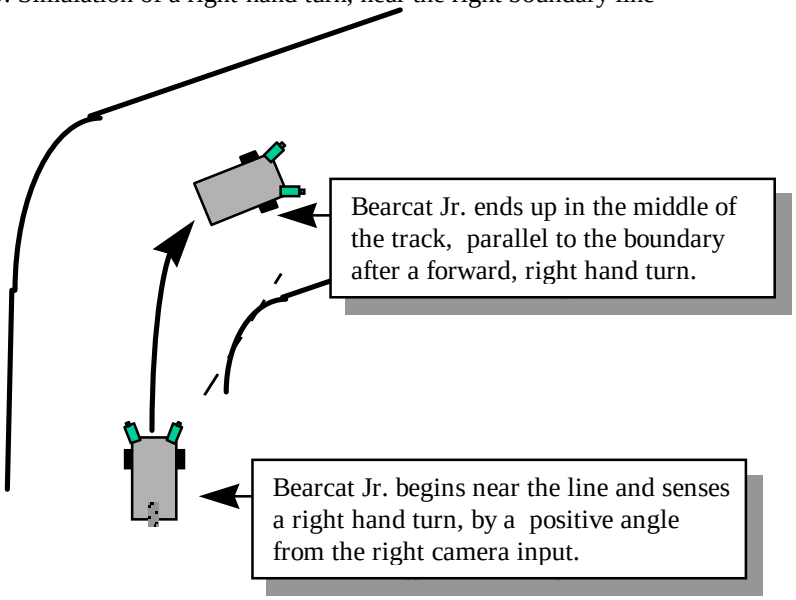
Table 3: Right camera control logic matrix

Right Camera Control

	Angle				
Distance	**Very Negative**	**Negative**	**Zero**	**Positive**	**Very Positive**
Near	**RSF LSB**	**RSF LSB**	**RF LSF**	**RSF LF**	**RSF LF**
Good	**RSF LSB**	**RSF LSB**	**RF LF**	**RSB LSF**	**RSB LSF**
Far	**RF LSF**	**RF LSF**	**RSF LF**	**RSB LSF**	**RSB LSF**

Table 3 represents the primary format of the logic. The distance input is located on the vertical axis, and the angle input is located on the horizontal axis. Based on the fuzzified values of the input, the controller selects the appropriate motor conditions listed in the matrix (RSF=right motor slow forward, and LSB=left motor slow backward, etc.) Since there is an overlap in the angle and distance membership functions, a combination of these rules, based on the aggregate method describe above, is used to select the velocity of each motor. This logic is demonstrated in the following examples:

1.) If the vehicle is right camera controlled, and the distance is near, and the angle is positive, the right motor speed is slow forward, and the left motor speed is forward. (These motor speed selections will produce a forward right turn.)
Figure 10: Simulation of a right-hand turn, near the right boundary line



2.) If the vehicle is right camera controlled, and the distance is near, and the angle is negative, the right motor speed is slow forward, and the left motor speed is slow backward. (These motor speed selections will produce a zero turning radius turn to the left.)

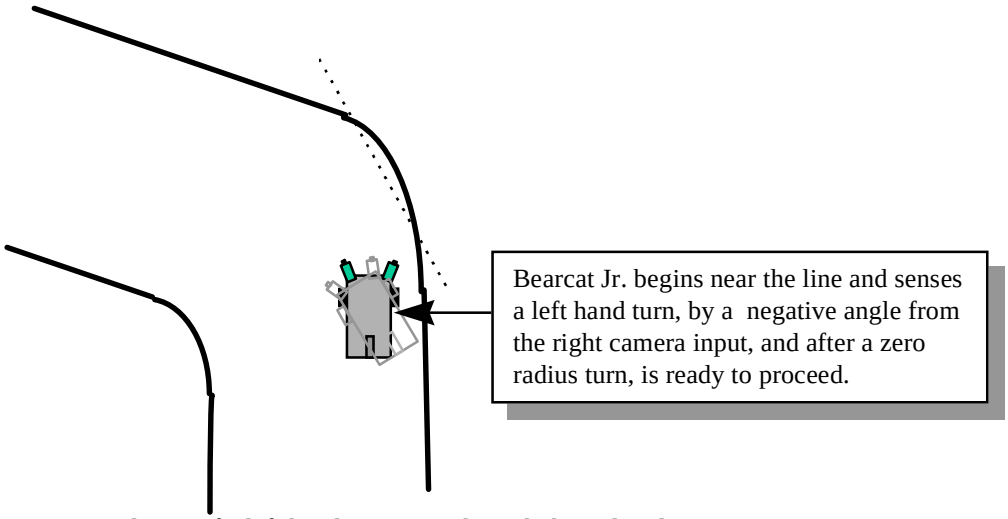


Figure 11: Simulation of a left-hand turn, near the right boundary line

With each iteration, the camera selection, distance, and angle information is updated, and the system selects new motor speeds based on the current information. This logic drives the trajectory of the cart to a position in the middle of the boundary lines, parallel to the lines, which is in the center of Table 3. At this position, both motor speeds are at forward velocity, propelling the vehicle forward at full speed.

Left Camera Control

Although the left camera control scheme is based upon the same fundamentals of the right camera control scheme, which is to bring the cart parallel to the boundary lines, and then to the center of the track, the motor speed selection criteria are different to account for the different position of the line relative to the Bearcat Jr. vehicle. The following schematic represents the Bearcat Jr. vehicle in a situation under left camera control.

Table 4: Left camera control logic matrix

Left Camera Control

	Angle				
Distance	**Very Negative**	**Negative**	**Zero**	**Positive**	**Very Positive**
Near	**RF LSF**	**RF LSF**	**RSF LF**	**RSB LSF**	**RSB LSF**
Good	**RSF LSB**	**RSF LSB**	**RF LF**	**RSB LSF**	**RSB LSF**
Far	**RSF LSB**	**RSF LSB**	**RF LSF**	**RSF LF**	**RSF LF**

As a direct comparison to the two right camera control examples shown in figures 10 and 11, the following situations represent similar conditions to situations described above. However, the position of the vehicle requires the vehicle to be under left camera control since it is nearer the left-hand boundary.

1.) If the vehicle is left camera controlled, and the distance is near, and the angle is positive, the right motor speed is slow backward, and the left motor speed is slow forward. (These motor speed selections will produce a forward right turn.)

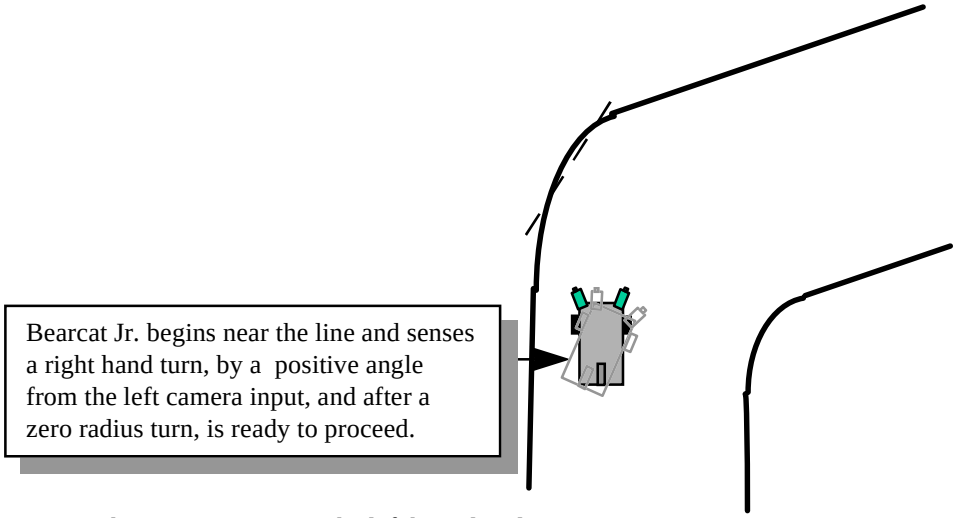


Figure 12: Right camera turn, near the left boundary line

2.) If the vehicle is left camera controlled, and the distance is near, and the angle is negative, the right motor speed is forward, and the left motor speed is slow forward. (These motor speed selections will produce a forward left-hand turn.)

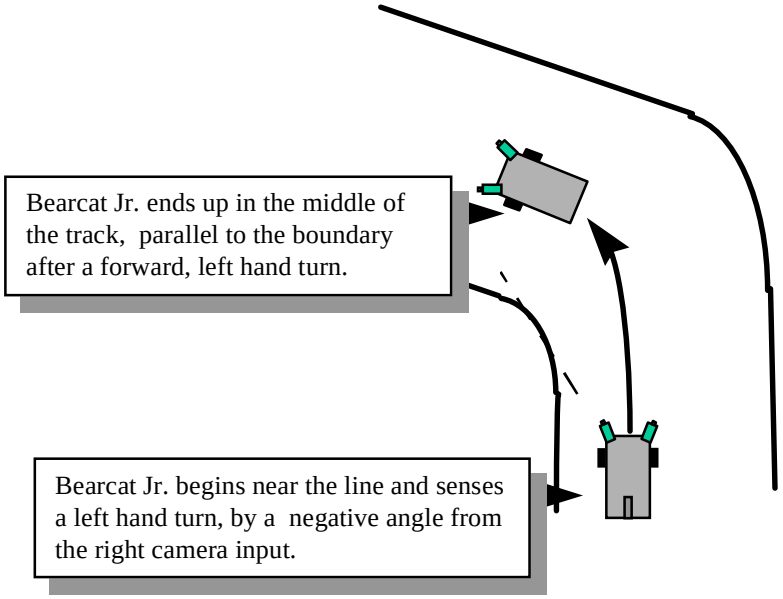


Figure 13: Simulation of left-hand turn, near the left boundary line

The C++ code developed for the control of Bearcat Jr. based on the left camera and right camera control logic is listed in Appendix B. This code computes the output motor velocity for the left and right motor, based on the inputs of the right and left camera, using the 5 principle steps of fuzzy logic.

5. System Evaluation

A Matlab simulation was developed to test the control logic, and the C++ coding of the fuzzy logic controller. The system, created within the Matlab Fuzzy Logic Toolbox, contains the same membership functions for the position and angular inputs, and the desired left and right motor outputs. Once the membership functions are identified within the toolbox, the control logic is simply programmed as linguistic rules relating the input to the

output. The Matlab interface does not allow combination (AND statements) in the output signal, so 2 statements are written for each block in the controller Matrix shown in Tables 3 and 4 above (one for the left motor signal and one for the right motor signal). A sample of the Matlab linguistic rules is shown in Appendix A.[4]

The system was tested, and debugged using the Matlab simulation as a benchmark. The following set of test points was selected randomly from the design space, representing several of the possible conditions that will be encountered by the Bearcat Jr. vehicle. Each test point was input into the Matlab model as well as the C++ controller code. Utilizing the output of the Matlab model, the controller code was debugged until the controller code output matched the output of the Matlab model, as shown in Table 7.

Table 7: System evaluation data test points

				Matlab Output		C++ Controller Output		
Left Camera Angle	**Left Camera Distance**	**Right Camera Angle**	**Right Camera Distance**	**Left Motor Speed**	**Right Motor Speed**	**Left Motor Speed**	**Right Motor Speed**	**Robot Motion**
10.8	3.9	-4.2	1.7	17.4	68.7	17.4	68.7	Forward Left Turn
22.5	5.1	24.8	3.3	40.0	-40.0	40.0	-40.0	Zero radius Right Turn
55.1	0.5	-57.4	0.4	-40.0	40.0	-40.0	40.0	Zero radius Left Turn
-87.5	4.4	-9.4	3.1	27.9	60.7	27.9	60.7	Forward Left Turn
16.9	0.9	-20.2	1.2	50.2	-19.7	50.2	-19.7	Right Turn
-1.0	3.4	34.1	4.4	52.0	74.8	52.0	74.8	Forward Left Turn
-9.7	3.8	14.1	1.9	65.8	54.2	65.8	54.2	Forward Right Turn
-29.3	1.1	-2.9	2.8	40.0	80.0	40.0	80.0	Forward Left Turn
76.9	3.5	-67.9	3.3	-17.3	51.3	-17.3	51.3	Left turn
14.4	5.9	9.0	2.3	59.9	43.4	59.9	43.4	Forward Right Turn

This sample data analysis verifies the accuracy of the application of the 5 principles of fuzzy logic. The code will now be integrated with the other vehicle systems enabling the vehicle to successfully navigate through the obstacle course.

6. Conclusions and Recommendations

A fuzzy logic control algorithm for the propulsion and steering of the Bearcat Jr. Autonomous vehicle has been developed and tested, and successfully applies the 5 principle steps of fuzzy logic. This controller forms the backbone of the Bearcat Jr. speed and steering control system.

Development of the vision and electrical system for the Bearcat Jr. vehicle continues. The input data requirements for the controller, which is the output of the vision system, have been communicated to the vision system development team. Once all systems have been developed and linked, the Bearcat Jr. vehicle will be ready for a final system test. This test will involve a final system tuning, selecting design parameters, and refining the membership functions to optimize vehicle speed and performance. Further development of the speed and steering control logic will center around obstacle sensing and obstacle avoidance. This project will expand the path navigation logic to utilize inputs from both the vision system, and from a sonar object sensing system. Motor speeds will be selected that will not only cause Bearcat Jr. to stay within the path boundaries, but also avoid the obstacles that it may encounter along the path.

References

1. Cao, Ming, Ernest Hall. “Fuzzy Logic Control for an Automated Guide Vehicle”, University of Cincinnati, 1997.
2. Galil Inc, "DMC-1000 Technical Reference Guide Ver 1.1", Sunnyvale, California 1993.
3. Hall , E.L. and B.C. Hall, "Robotics: A User-Friendly Introduction", Holt, Rinehart, and Winston, New York, NY, 1985.
4. Jang, J.S. Roger, Ned Gulley. Matlab Fuzzy Logic Toolbox Users Guide. © 1995, The Mathworks Inc. Natwick, MA.
5. Kolli, Kalyan and Ernest L. Hall. "Steering Control System for a Mobile Robot", Proceedings of SPIE - The International Society for Optical Engineering. v 3208, Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, USA, 1997, pp. 162-169.
6. Kolli, Kaylan, Sreeram Mallikarjun, Krishnamohan Kola and Ernest L. Hall, "Speed Control for a Mobile Robot", Proceedings of SPIE - The International Society for Optical Engineering. v 3208, Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, USA, 1997, pp. 104-110.
7. Leonard, Naomi Ehrich, William S. Levine. Using Matlab to Analyze and Design Control Systems. © 1995, Addison-Wesley, New York, New York.
8. Matthews, Bradley O., Michael A. Ruthemeyer, David Perdue, Ernest L. Hall, "Development of a mobile robot for the 1995 AUVS competition", Proceedings of SPIE - The International Society for Optical Engineering, v 2591, Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, USA, 1995, pp. 194-201.
9. McNeil , Daniel and Paul Freiberger. Fuzzy Logic. © 1993, Simon and Schuster, New York, NY.
10. Reliance Electric, "Electro-Craft BDC-12 Instruction Manual", Eden Prairie, Minnesota 1993.
11. Rogers, Michael. “The Future Looks Fuzzy”, Newsweek, May 26, 1990, pp.46-47.
12. Samu ,Tayib, Nikhal Kelkar, David Perdue, Michael A. Ruthemeyer, Bradley O. Matthews, Ernest L. Hall, "Line following using a two camera guidance system for a mobile robot", Proceedings of SPIE - the International Society for Optical Engineering, v 2904, 1996, pp. 290-297.
13. Sowell, Thomas. Fuzzy Logic for ‘Just Plain Folks’. Lubbock, Texas, © 1997.
14. Waite, Mitchell, Stephen Prata. C Primer Plus. © 1993, Sams Publishing, Carmell, Indiana.
15. Zadeh, Lofti. “The Concept of a Linguistic Variable and its Application to Approximate Reasoning.” Information Sciences, 8 1975, pp.199-249.
16. Zadeh, Lofti. “Making Computers Think Like People,” IEEE Spectrum, August 1984, pp. 26-32.

Appendix A

Matlab Fuzzy Logic Interface System Rules – Right Camera Control

1.) If the distance is near and the angle is very negative, then the right motor speed is slow forward.
2.) If the distance is near and the angle is very negative, then the left motor speed is slow backward.
3.) If the distance is near and the angle is negative, then the right motor speed is slow forward.
4.) If the distance is near and the angle is negative, then the left motor speed is slow backward.
5.) If the distance is near and the angle is zero, then the right motor speed is forward.
6.) If the distance is near and the angle is zero, then the left motor speed is slow forward.
7.) If the distance is near and the angle is positive, then the right motor speed is slow forward.
8.) If the distance is near and the angle is positive, then the left motor speed is forward.
9.) If the distance is near and the angle is very positive, then the right motor speed is slow forward.
10.) If the distance is near and the angle is very positive, then the left motor speed is forward.
11.) If the distance is good and the angle is very negative, then the right motor speed is slow forward.
12.) If the distance is good and the angle is very negative, then the left motor speed is slow backward.
13.) If the distance is good and the angle is negative, then the right motor speed is slow forward.
14.) If the distance is good and the angle is negative, then the left motor speed is slow backward.
15.) If the distance is good and the angle is zero, then the right motor speed is forward.
16.) If the distance is good and the angle is zero, then the left motor speed is forward.
17.) If the distance is good and the angle is positive, then the right motor speed is slow backward.
18.) If the distance is good and the angle is positive, then the left motor speed is slow forward.
19.) If the distance is good and the angle is very positive, then the right motor speed is slow backward.
20.) If the distance is good and the angle is very positive, then the left motor speed is slow forward.
21.) If the distance is near and the angle is very negative, then the right motor speed is forward.
22.) If the distance is far and the angle is very negative, then the left motor speed is slow forward.
23.) If the distance is far and the angle is negative, then the right motor speed is forward.
24.) If the distance is far and the angle is negative, then the left motor speed is slow forward.
25.) If the distance is far and the angle is zero, then the right motor speed is slow forward.
26.) If the distance is far and the angle is zero, then the left motor speed is forward.
27.) If the distance is far and the angle is positive, then the right motor speed is slow backward.
28.) If the distance is far and the angle is positive, then the left motor speed is slow forward.
29.) If the distance is far and the angle is very positive, then the right motor speed is slow backward.
30.) If the distance is far and the angle is very positive, then the left motor speed is slow forward.

Appendix B

```
#include <stdio.h>

struct camera {
        double angle;
        double distance;
};

struct fuzzy {
        double rmotor;
        double lmotor;
};

struct camera rcamera(void)
{
/*
        This function computes the angle and distance from the right camera
*/
        struct camera rc;
        float ang ,dist;
        printf("\nEnter right camera angle ");
        scanf("%f",&ang);
        rc.angle=ang;
        printf("\nEnter right camera distance ");
        scanf("%f",&dist);
        rc.distance=dist;
/*      printf(" %f, %f . \n", rc.angle, rc.distance);*/
        return rc;
};

struct camera lcamera(void)
{
/*
        This function computes the angle and distance from the left camera
*/
        struct camera lc;
        float lang, ldist;
        printf("\nEnter left camera angle ");
        scanf("%f",&lang);
        lc.angle=lang;
        printf("\nEnter left camera distance ");
        scanf("%f",&ldist);
        lc.distance=ldist;
/*      printf(" %f, %f . \n", lc.angle, lc.distance); */
        return lc;
};

struct fuzzy rcamctl(struct camera rcamin)
{
        int anisvneg=0;
        int anisneg=0;
        int aniszero=0;
        int anispos=0;
        int anisvpos=0;
```

```
        double vnegangl=-90;
        double vnegangr=-40;
        double negangl=-40;
        double negangr=0;
        double zerangl=-20;
        double zerangr=20;
        double posangl=0;
        double posangr=40;
        double vposangl=40;
        double vposangr=90;

        double neardistl=0;
        double neardistr=3;
        double gooddistl=2;
        double gooddistr=4;
        double fardistl=3;
        double fardistr=6;

        int distisnear=0;
        int distisgood=0;
        int distisfar=0;

        double lmbackl=-100;
        double lmbackr=-60;
        double lmslbkl=-60;
        double lmslbkr=-20;
        double lmslowl=-20;
        double lmslowr=20;
        double lmslfwdl=20;
        double lmslfwdr=60;
        double lmfwdl=60;
        double lmfwdr=100;

        double rmbackl=-100;
        double rmbackr=-60;
        double rmslbkl=-60;
        double rmslbkr=-20;
        double rmslowl=-20;
        double rmslowr=20;
        double rmslfwdl=20;
        double rmslfwdr=60;
        double rmfwdl=60;
        double rmfwdr=100;

        double angle=rcamin.angle;
        double distance=rcamin.distance;
        struct fuzzy out;

/*
        Test for set membership
        for angle parameter
*/
        if(angle < vnegangr) anisvneg=1;
        else if(angle <negangr) anisneg=1;
        else if(angle <posangr) anispos=1;
```

```
        else anisvpos=1;
        if(angle > zerangl && angle <zerangr ) aniszero=1;
/*
        Test for set membership
        for distance parameter
*/
        if(distance < neardistr) distisnear=1;
        else if (distance > fardistl) distisfar=1;
        if(distance > gooddistl && distance < gooddistr) distisgood=1;
        printf("anisvneg= %d, anisneg= %d, anispos= %d \n",anisvneg, anisneg, anispos);
        printf("anisvpos= %d, aniszero= %d\n",anisvpos, aniszero);
        printf("distisnear= %d, distisfar= %d, distisgood %d\n",distisnear, distisfar,distisgood);

/*
        Apply the fuzzy logic rules
*/
        double rareasum=0, lareasum=0;
        double rmomentsum=0, lmomentsum=0;
        double trimf(double value, double tri1, double tri2);
        double mf1, mf2, mf;
        double areacalc(double val1, double val2, double val3);
        double rareavalue, rcentvalue, lareavalue, lcentvalue;
        double centcalc(double ll, double rr);
/*
         If (angle is Zero) and (distance is good) then (rmotor is Forward) and
         (lmotor if forward)

*/
        if(aniszero==1 && distisgood==1)
        {
                mf1=trimf(angle, zerangl, zerangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                printf("The final mf value is %2.2f \n",mf);
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
         If (angle is Zero) and (distance is far) then (rmotor is Slow Forward) and
         (lmotor if forward)

*/
        if(aniszero==1 && distisfar==1)
        {
                mf1=trimf(angle, zerangl, zerangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
```

```
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Zero) and (distance is near) then (rmotor is Forward) and
        (lmotor if Slow forward)

*/
        if(aniszero==1 && distisnear==1)
        {
                mf1=trimf(angle, zerangl, zerangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Positive) and (distance is good) then (rmotor is Slow Back) and
        (lmotor if Slow Forward)

*/
        if(anispos==1 && distisgood==1)
        {
                mf1=trimf(angle, posangl, posangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Positive) and (distance is near) then
        (rmotor is Slow Forward) and (Lmotor is Forward)
```

```
*/
        if(anispos==1  && distisnear==1)
        {
                mf1=trimf(angle, posangl, posangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
         If (angle is Positive) and (distance is far)
         then (rmotor is Slow Back) and (Lmotor is Slow Forward)

*/
        if(anispos==1  && distisfar==1)
        {
                mf1=trimf(angle, posangl, posangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
         If (angle is Negative) and (distance is far) then (rmotor is Forward) and
         (Lmotor is Slow Forward)

*/
        if(anisneg==1  && distisfar==1)
        {
                mf1=trimf(angle, negangl, negangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
```

```
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Negative) and (distance is good)
        then (rmotor is Slow Forward) and  (Lmotor is Slow Back)

*/
        if((anisneg==1) && distisgood==1)
        {
                mf1=trimf(angle, negangl, negangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Negative) and (distance is near) then (rmotor is Slow Forward) and
        (Lmotor is Slow Back)

*/
        if(anisneg==1 && distisnear==1)
        {
                mf1=trimf(angle, negangl, negangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Positive) and (distance is good) then (rmotor is Slow Back) and
        (lmotor if Slow Forward)

*/
        if(anisvpos==1 && distisgood==1)
        {
                mf1=trimf(angle, vposangl, vposangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
```

```
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Positive) and (distance is near) then
        (rmotor is Slow Forward) and (Lmotor is Forward)

*/
        if(anisvpos==1  && distisnear==1)
        {
                mf1=trimf(angle, vposangl, vposangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Positive) and (distance is far)
        then (rmotor is Slow Back) and (Lmotor is Slow Forward)

*/
        if(anisvpos==1  && distisfar==1)
        {
                mf1=trimf(angle, vposangl, vposangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Negative) and (distance is far) then (rmotor is Forward) and
        (Lmotor is Slow Forward)
```

```
*/
        if(anisvneg==1  && distisfar==1)
        {
                mf1=trimf(angle, vnegangl, vnegangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
         If (angle is Very Negative) and (distance is good)
         then (rmotor is Slow Forward) and  (Lmotor is Slow Back)

*/
        if((anisvneg==1) && distisgood==1)
        {
                mf1=trimf(angle, vnegangl, vnegangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
         If (angle is Very Negative) and (distance is near) then (rmotor is Slow Forward) and
         (Lmotor is Slow Back)

*/
        if(anisvneg==1 && distisnear==1)
        {
                mf1=trimf(angle, vnegangl, vnegangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
```

```
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }


/*
        Defuzzify system
*/
        if (rareasum==0 | lareasum==0)
        {
                printf("ERROR DIVISION BY 0  STOPPING ROBOT");
                out.rmotor=0;
                out.lmotor=0;
        }
        else
        {       out.rmotor=rmomentsum/rareasum;
                out.lmotor=lmomentsum/lareasum;
                printf("The fuzzy output for the system right motor is %2.2f\n",out.rmotor);
                printf("The fuzzy output for the system left motor is %2.2f\n",out.lmotor);
        }
        return out;
}


struct fuzzy lcamctl(struct camera lcamin)
{
        int anisvneg=0;
        int anisneg=0;
        int aniszero=0;
        int anispos=0;
        int anisvpos=0;

        double vnegangl=-90;
        double vnegangr=-40;
        double negangl=-40;
        double negangr=0;
        double zerangl=-20;
        double zerangr=20;
        double posangl=0;
        double posangr=40;
        double vposangl=40;
        double vposangr=90;

        double neardistl=0;
        double neardistr=3;
        double gooddistl=2;
        double gooddistr=4;
        double fardistl=3;
        double fardistr=6;

        int distisnear=0;
        int distisgood=0;
        int distisfar=0;

        double lmbackl=-100;
        double lmbackr=-60;
        double lmslbkl=-60;
```

```
        double lmslbkr=-20;
        double lmslowl=-20;
        double lmslowr=20;
        double lmslfwdl=20;
        double lmslfwdr=60;
        double lmfwdl=60;
        double lmfwdr=100;

        double rmbackl=-100;
        double rmbackr=-60;
        double rmslbkl=-60;
        double rmslbkr=-20;
        double rmslowl=-20;
        double rmslowr=20;
        double rmslfwdl=20;
        double rmslfwdr=60;
        double rmfwdl=60;
        double rmfwdr=100;

        double angle=lcamin.angle;
        double distance=lcamin.distance;
        struct fuzzy out;

/*
        Test for set membership
        for angle parameter
*/
        if(angle < vnegangr) anisvneg=1;
        else if(angle <negangr) anisneg=1;
        else if(angle <posangr) anispos=1;
        else anisvpos=1;
        if(angle > zerangl && angle <zerangr ) aniszero=1;
/*
        Test for set membership
        for distance parameter
*/
        if(distance < neardistr) distisnear=1;
        else if (distance > fardistl) distisfar=1;
        if(distance > gooddistl && distance < gooddistr) distisgood=1;
        printf("anisvneg= %d, anisneg= %d, anispos= %d \n",anisvneg, anisneg, anispos);
        printf("anisvpos= %d, aniszero= %d\n",anisvpos, aniszero);
        printf("distisnear= %d, distisfar= %d, distisgood %d\n",distisnear, distisfar,distisgood);

/*
        Apply the fuzzy logic rules
*/
        double rareasum=0, lareasum=0;
        double rmomentsum=0, lmomentsum=0;
        double trimf(double value, double tri1, double tri2);
        double mf1, mf2, mf;
        double areacalc(double val1, double val2, double val3);
        double rareavalue, rcentvalue, lareavalue, lcentvalue;
        double centcalc(double ll, double rr);
/*
        If (angle is Zero) and (distance is good) then (rmotor is Forward) and
        (lmotor if forward)
```

```
*/
        if(aniszero==1 && distisgood==1)
        {
                mf1=trimf(angle, zerangl, zerangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                printf("The final mf value is %2.2f \n",mf);
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
         If (angle is Zero) and (distance is near) then (rmotor is Slow Forward) and
         (lmotor if forward)

*/
        if(aniszero==1 && distisnear==1)
        {
                mf1=trimf(angle, zerangl, zerangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
         If (angle is Zero) and (distance is far) then (rmotor is Forward) and
         (lmotor if Slow forward)

*/
        if(aniszero==1 && distisfar==1)
        {
                mf1=trimf(angle, zerangl, zerangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
```

```
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Positive) and (distance is good) then (rmotor is Slow Back) and
        (lmotor if Slow Forward)

*/
        if(anispos==1 && distisgood==1)
        {
                mf1=trimf(angle, posangl, posangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Positive) and (distance is far) then
        (rmotor is Slow Forward) and (Lmotor is Forward)

*/
        if(anispos==1  && distisfar==1)
        {
                mf1=trimf(angle, posangl, posangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Positive) and (distance is near)
        then (rmotor is Slow Back) and (Lmotor is Slow Forward)

*/
        if(anispos==1  && distisnear==1)
        {
                mf1=trimf(angle, posangl, posangr);
                mf2=trimf(distance, neardistl, neardistr);
```

```
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Negative) and (distance is near) then (rmotor is Forward) and
        (Lmotor is Slow Forward)

*/
        if(anisneg==1  && distisnear==1)
        {
                mf1=trimf(angle, negangl, negangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Negative) and (distance is good)
        then (rmotor is Slow Forward) and  (Lmotor is Slow Back)

*/
        if((anisneg==1) && distisgood==1)
        {
                mf1=trimf(angle, negangl, negangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Negative) and (distance is far) then (rmotor is Slow Forward) and
```

```
        (Lmotor is Slow Back)

*/
        if(anisneg==1 && distisfar==1)
        {
                mf1=trimf(angle, negangl, negangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Positive) and (distance is good) then (rmotor is Slow Back) and
        (lmotor if Slow Forward)

*/
        if(anisvpos==1 && distisgood==1)
        {
                mf1=trimf(angle, vposangl, vposangr);
                mf2=trimf(distance, gooddistl, gooddistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Positive) and (distance is far) then
        (rmotor is Slow Forward) and (Lmotor is Forward)

*/
        if(anisvpos==1  && distisfar==1)
        {
                mf1=trimf(angle, vposangl, vposangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmfwdl,lmfwdr);
```

```
                lcentvalue=centcalc(lmfwdl, lmfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Positive) and (distance is near)
        then (rmotor is Slow Back) and (Lmotor is Slow Forward)

*/
        if(anisvpos==1  && distisnear==1)
        {
                mf1=trimf(angle, vposangl, vposangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslbkl,rmslbkr);
                rcentvalue=centcalc(rmslbkl, rmslbkr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Negative) and (distance is near) then (rmotor is Forward) and
        (Lmotor is Slow Forward)

*/
        if(anisvneg==1  && distisnear==1)
        {
                mf1=trimf(angle, vnegangl, vnegangr);
                mf2=trimf(distance, neardistl, neardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmfwdl,rmfwdr);
                rcentvalue=centcalc(rmfwdl, rmfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslfwdl,lmslfwdr);
                lcentvalue=centcalc(lmslfwdl, lmslfwdr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Negative) and (distance is good)
        then (rmotor is Slow Forward) and  (Lmotor is Slow Back)

*/
        if((anisvneg==1) && distisgood==1)
        {
                mf1=trimf(angle, vnegangl, vnegangr);
                mf2=trimf(distance, gooddistl, gooddistr);
```

```
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }

/*
        If (angle is Very Negative) and (distance is far) then (rmotor is Slow Forward) and
        (Lmotor is Slow Back)

*/
        if(anisvneg==1 && distisfar==1)
        {
                mf1=trimf(angle, vnegangl, vnegangr);
                mf2=trimf(distance, fardistl, fardistr);
                if(mf1<mf2) mf=mf1;
                else mf=mf2;
                rareavalue=areacalc(mf,rmslfwdl,rmslfwdr);
                rcentvalue=centcalc(rmslfwdl, rmslfwdr);
                rareasum=rareasum+rareavalue;
                rmomentsum=rmomentsum+rcentvalue*rareavalue;
                lareavalue=areacalc(mf,lmslbkl,lmslbkr);
                lcentvalue=centcalc(lmslbkl, lmslbkr);
                lareasum=lareasum+lareavalue;
                lmomentsum=lmomentsum+lcentvalue*lareavalue;
        }


/*
        Defuzzify system
*/
        if (rareasum==0 | lareasum==0)
        {
                printf("ERROR DIVISION BY 0  STOPPING ROBOT");
                out.rmotor=0;
                out.lmotor=0;
        }
        else
        {       out.rmotor=rmomentsum/rareasum;
                out.lmotor=lmomentsum/lareasum;
                printf("The fuzzy output for the system right motor is %2.2f\n",out.rmotor);
                printf("The fuzzy output for the system left motor is %2.2f\n",out.lmotor);
        }
        return out;
}

double trimf(double val, double a, double b)
{
                double mf;
                if (val < a) mf=0;
```

```
                else if (val < (b+a)/2) mf=2*(val-a)/(b-a);
                else if (val >b) mf=0;
                else if (val >= (b+a)/2) mf=2*(b-val)/(b-a);

                printf("\n The membership function value is %2.2f\n",mf);
                return mf;
}

double areacalc(double mem,double l,double r)
{
        double area;
        area=.5*mem*(l-r)*(2-mem);
        return area;
}

double centcalc(double l, double r)
{
        double centroid;
        centroid=(l+r)/2;
        return centroid;
}


struct camera lcam;
struct camera rcam;
struct fuzzy spdout;
int go=1;

int main(void)
{
        while(go==1)
        {
        lcam=lcamera();
/*
        printf("The left camera parameters are");
        printf(" %f, %f . \n", lcam.angle, lcam.distance);
*/
        rcam=rcamera();
/*
        printf("The right camera parameters are");
        printf(" %f, %f . \n", rcam.angle, rcam.distance);
*/
        if(lcam.distance < rcam.distance)
        {
                printf("\n\nLeft Camera Control\n\n");
                spdout=lcamctl(lcam);
        }
        else
        {
                printf("\n\nRight Camera Control\n\n");
                spdout=rcamctl(rcam);
        }

        speedCARl((long int)spdout.lmotor);
        speedCARr((long int)spdout.rmotor);
        printf("LM %2.2f, RM %2.2f\n",spdout.lmotor,spdout.rmotor);
```

```
                }
                return 0;

};
```