The FuzzBug Mobile Robot

Sean Graves John Mollenhauer Marge Skubic

August 30, 1993

I. Introduction

This document describes the design and implementation of an autonomous mobile platform which uses fuzzy logic for navigation and collision avoidance. Specific topics discussed include the motivation, rationale, development, and performance evaluation of the mobile platform we call the FuzzBug.

II. Motivation and Description of the Problem

The goal of this project is to build a self-contained, mobile robot which can be controlled by a small, on-board processor. The primary objective is to use fuzzy logic for fusing noisy sensor data to perform navigation in unknown environments. The robot must track a target (an LED beacon) and at the same time avoid collisions with obstacles. The system must also function properly in a changing environment. The latter aspect suggests the flavor of reactive control, where the mobile robot appears to "react" to its changing environment.

Part of the challenge is to use inexpensive components, which means a very small processor, limited memory, and noisy sensors. The project tests the limits of the cost vs. utility tradeoff that the fuzzy logic approach can provide. That is, just how cost effective can a fuzzy logic controller be? In other words, what is the least amount of memory and computing power needed for a fuzzy logic controller to perform reasonable navigation of a mobile robot in a real-world, dynamic environment?

A major motivation for the use of fuzzy logic in this project is that fuzzy logic can combine data from various sources. Our robot design consists of three primary sensor types, all of which are noisy and may present conflicting information. It is difficult to design traditional (crisp) controllers that can meet design goals such as ours, given the low quality of the sensors we are using.

Another motivation for using fuzzy logic in this project is that our environment is constantly changing. In fact, as the robot moves around in its environnment, different ambient light levels are constantly encountered. Because the sensors are sensitive to ambient light, a crisp representation of whether a obstacle is detected or not could often be incorrect because the boundary value is constantly changing. Fuzzy logic allows the boundary to be gradual instead of a sharp line. This has the effect of smoothing out the effects of the variations.

III. Related Work

Song and Tai [1] proposed a fuzzy logic navigation method for a mobile robot in an unknown environment. The mobile robot used two independent driving wheels and one free wheel. Steering was performed by commanding different velocities to the two driving wheels. The robot contained six ultrasonic sensors, which were used to measure distances to immediate obstacles. Data from the sensors was used to determine the turning direction, which was translated to command velocities for the driving wheels.

The six ultrasonic sensors were used in pairs. Two sensors were directed forward, two were directed towards the left, and two were directed towards the right. Data from the left pair and the front pair was used for the Left Fuzzy Logic Controller (LFLC), which controlled a left turn. Likewise, data from the right pair and the front pair was used for the Right Fuzzy Logic Controller (RFLC), which controlled a right turn. If the target direction was straight in front, then either the LFLC or the RFLC would be used, depending on which side was nearer to an obstacle. Each FLC used four inputs (the ultrasonic sensors), generated two outputs (the command velocities for the driving wheels), and used 81 rules.

The overall architecture used a combination of fuzzy logic and non-fuzzy logic navigation methods. A higher level navigation controller was used to check sensor data, target location, and robot location, and then determine which control strategy to use. In this way, control could be switched automatically between the LFLC, the RFLC, and other non-fuzzy control strategies. The non-fuzzy controllers were used under special conditions, such as maneuvering in a narrow corridor.

The navigation method proposed by Song and Tai has some limitations. First, it would be too complex to implement on a very small computer with limited memory (two FLC's with 81 rules each!). In addition, the method does not join the target tracking operation with the obstacle avoidance operation. Fuzzy logic is used only for the obstacle avoidance function. A higher level controller has to "point" the robot at the target. Also, the results supplied in the paper are simulation results only. The authors acknowledge that the simulation assumes perfect conditions. That is, the effects of noisy sensors, imperfect vehicle dynamics, and wheel inertia are ignored.

Another fuzzy logic navigation method was proposed by Yen and Pfluger [2]. Their method also assumes six ultrasonic sensors mounted on a mobile robot, used to provide distance information to obstacles. A desired target direction is combined with data from the ultrasonic sensors to determine an acceptable turning angle. The fuzzy logic approach uses the following steps:

1. The desired direction is used to calculate the turn angle, which is then fuzzified (i.e., broadened).
2. Undesirable turning angles are determined from the sensor data, using fuzzy sets.
3. The desired turn angle and the undesirable turn angles are combined.

4. The result is defuzzified to determine a commandable turning angle.

These two approaches are examples of using fuzzy logic for navigation control of a mobile robot. Neither strategy could be used as is for our project, because of hardware limitations (especially sensors and memory). However, each can provide some insight into the problem and an acceptable solution.

One conclusion is that the target tracking operation should probably be merged with the obstacle avoidance operation. If these two functions can be merged using fuzzy logic, the resulting controller will probably be simpler and more efficient than the sum of the two separate operations.

IV. Design Rationale

A. *Hardware Components*

In keeping with our objective to use inexpensive components, the structure of the FuzzBug is designed using Lego Technic parts. Lego pieces can be easily configured and modified as necessary. Adequate wheels, gears, and motors are available to provide a controllable, mobile base. For our project, we use two standard Lego Technic motors, which draw a maximum of 600 mA of current.

For on-board processing power, Mini Boards are used. The Mini Board is ideal for this project, as it is small (2 by 3 inches) and has I/O ports as well as a microcontroller chip. The Mini Board contains a 68HC811 microcontroller, 4 motor control ports, 8 digital input ports, and 8 analog input ports. The 68HC811 contains 256 bytes of RAM and 2048 bytes of EEPROM. Each Mini Board draws approximately 100 mA of current. The Mini Board was designed by Fred Martin of MIT for use in mobile robot competitions and anywhere else that a small powerful computer is needed. Its design has been placed into the public domain via the comp.robotics Internet newsgroup.

A variety of inexpensive sensors are used to provide input data. Infrared (IR) emitter/detector pairs are used to detect the presence of obstacles. These IR sensors are analog devices which can detect the amount of modulated emitted light reflected by an object. Therefore, the value given by digitizing these analog signals can give proximity information. The IR sensors are interfaced to the Mini Board via a custom circuit of our design which provides emitter modulation, detector amplification, bandpass filtering, and peak detection. This circuit draws 150 mA of current.

Another type of IR detector is used to track the IR LED beacon target. These sensors detect infrared light that is modulated at 40 KHz, which is the type of signal emitted by common infrared remote controls. This enables us to use any infrared remote as a beacon. The beacon detectors provide digital input data to the fuzzy inference unit (i.e., 0 or 1– the beacon is either visible or not visible to that particular sensor).

Yet another type of IR device is used to detect motor stalls. A TRW OPB-5447 infrared reflectance sensor is placed very close to the spokes of one of the drive wheels. As the spokes of the wheel pass in front of the sensor, pulses of IR light are detected. The sensor's output, although analog, varies enough to enable us to tie it directly to the 68HC811's pulse accumulator pin. This port

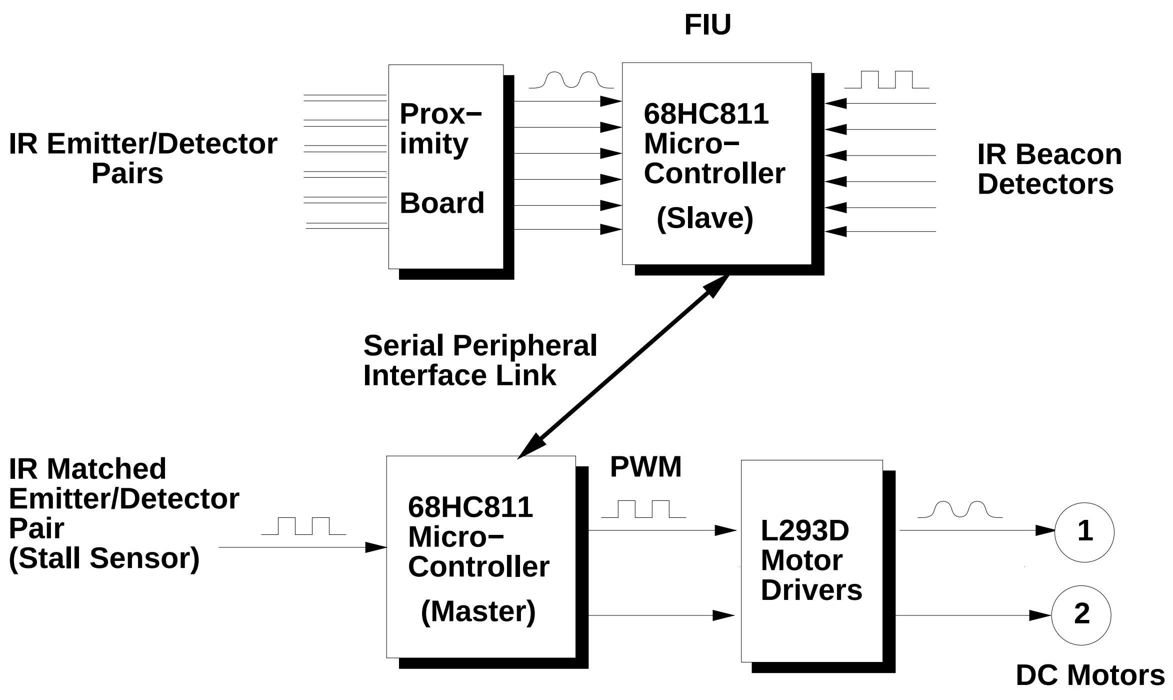
FIU
IR Emitter/Detector Pairs
Prox-imity Board
68HC811 Micro-Controller (Slave)
IR Beacon Detectors
Serial Peripheral Interface Link
IR Matched Emitter/Detector Pair (Stall Sensor)
68HC811 Micro-Controller (Master)
PWM
L293D Motor Drivers
1
2
DC Motors

Figure 1: Hardware connections of the FuzzBug controller

automatically counts voltage pulses, and makes that count available in a register. By periodically checking this register, we can verify that the wheels are moving when they should be.

Each of the three main boards on the FuzzBug (two Mini Boards and the IR board) have separate LM2931Z voltage regulators to provide 5 volts. The LM2931Z was designed for battery powered and automotive applications, and is extremely robust.

Figure 1 illustrates the hardware configuration of the fuzzbug. Figures 2 and 3 show the physical layout of the hardware components.

B. Software Design Rationale

We considered a number of different schemes of inferencing for our controller. A two-level system was eventually chosen because it saves memory, and simplifies the rule structure by reusing rules. By using the same set of rules and FIU code, we only needed nine rules to fuse data from 12 sensors. A single level FIU would have required many more rules, and tuning the controller would have been very difficult.

Another advantage of the multi-level system is that it allows modular testing. Because of the simplicity of the first level, we were able to complete testing that level of the FIU very quickly. Then, we evaluated a number of different techniques of combining the weights into a single crisp value. With only one level, this type of interated testing and analysis would have been impossible. For more information on the implementation details, including membership functions, rules, and

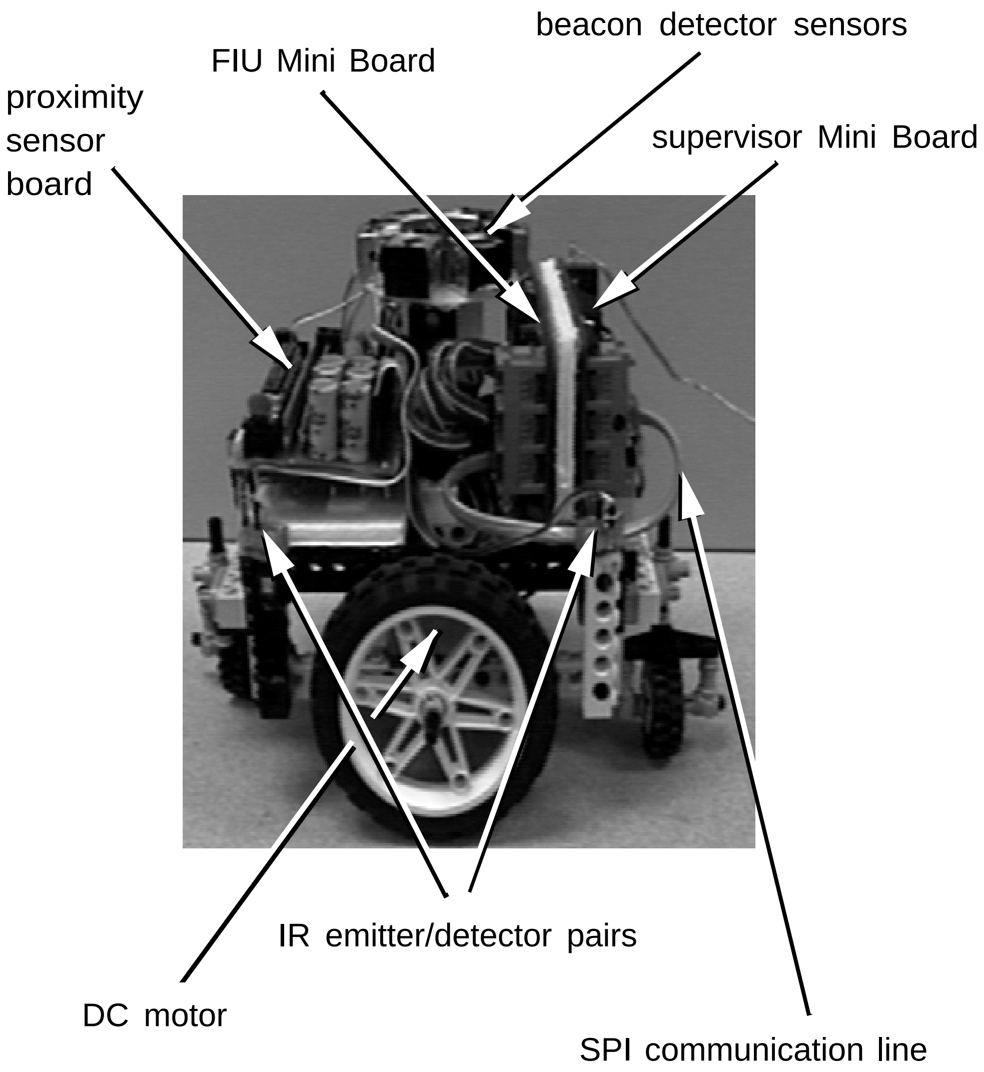

Figure 2: Side view of the FuzzBug, showing hardware component placement

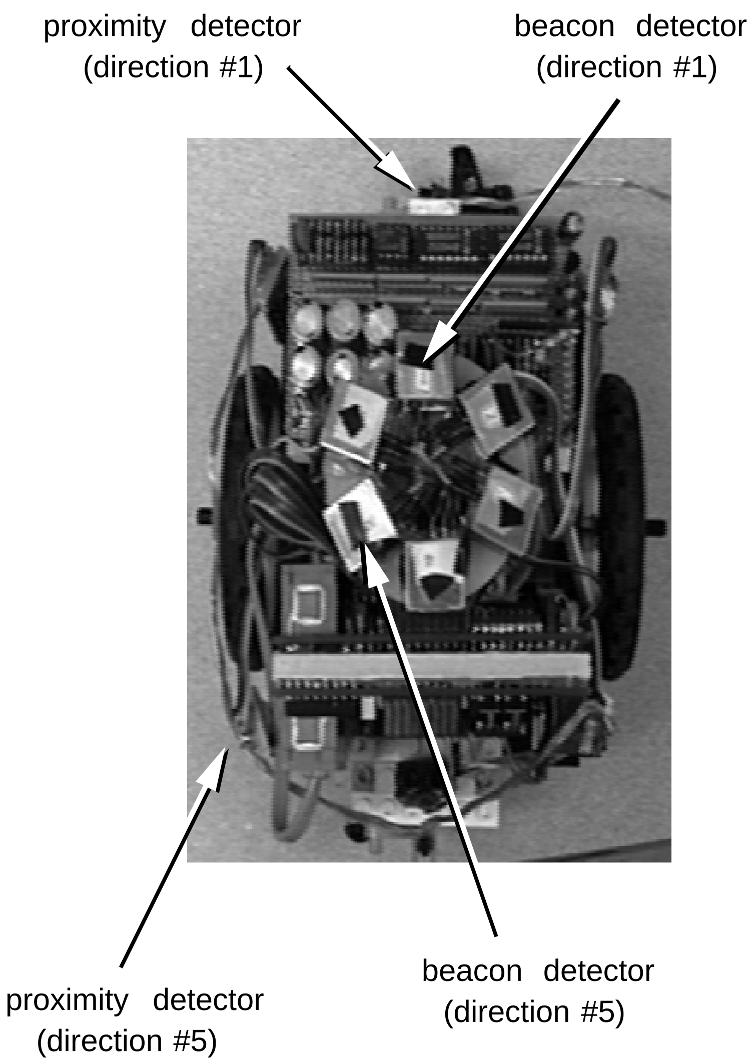
proximity detector (direction #1)
beacon detector (direction #1)
proximity detector (direction #5)
beacon detector (direction #5)

Figure 3: Top view of the FuzzBug, showing hardware component placement

defuzzification, refer to the next section.

Our control system utilizes two loosely-coupled processors which communicate via a synchronous serial link. This parallel processing scheme allows us to run our FIU at a different rate than our motor control code. We chose to separate the FIU from the supervisory code for the following reasons. First of all, it allows the processes to run at different rates. Also, this separation gave each process the full memory space of the Mini Board, and allowed us to save extra memory by customizing the runtime libraries and interrupt service routines to match the task. This approach is also more modular, and allowed testing of the components separately.

Our initial design of the fuzzy controller was to utilize FIDE to generate an FIU for the 6811 architecture. This technique has a number of problems. The size of the generated code is fairly large – for example, our test FIU took approximately 1K, which was half the available memory of our Mini Board. Also, FIDE only supports the TVFI defuzzification method for the 6811. Although our finished system uses TVFI, we initially wanted to use the Mamdani method. Finally, the FIDE FIU did not seem to work correctly despite our best efforts to debug it. Therefore, we decided to write our own FIU. This gives us a small, efficient module which we can readily modify. Also, we could not have easily implemented our multi-level inference method using FIDE.

V. Software Implementation

The FuzzBug software is distributed across two Mini Boards. Figure 4 shows the relationships between the software modules and the flow of data between them. The FIU Mini Board contains the Fuzzy Inference Units (FIU), a module to combine the outputs from the FIUs, and modules to read the beacon and proximity sensors. The supervisor Mini Board contains the control software. This board requests data from the FIU Mini Board, thus earning its name as the supervisor Mini Board.

A. FIU Mini Board

The purpose of the FIU Mini Board is to read the beacon and proximity sensors, fuzzify these inputs, execute a set of fuzzy inference rules, and then defuzzify the results. Figure 4 shows six separate FIUs. However, due to severe memory limitations (only 2K of memory), this is not how the system was actually implemented. Each FIU in the figure is identical, so instead of six FIUs operating in parallel, we have one FIU called six times, once for each sensor direction. The basic operation, from sensors through the Combine Weights module is as follows.

For each sensor direction the following occurs. The analog proximity sensor is read, with a result in the range of 0 to 255. (The lower the reading from these sensors, the more open the area around that sensor.) The digital beacon detection sensor is read, with a result of either 0 (no beacon detected) or 1 (beacon detected). These two readings are then fed into the FIU. The input data membership functions used by the FIU are shown in Figure 5. The proximity sensor has five fuzzy subsets to its domain. A low (L) reading indicates a clear path in the direction of the sensor, and a high (H) reading indicates an obstacle. The subsets between (ML = Medium Low, M = Medium,

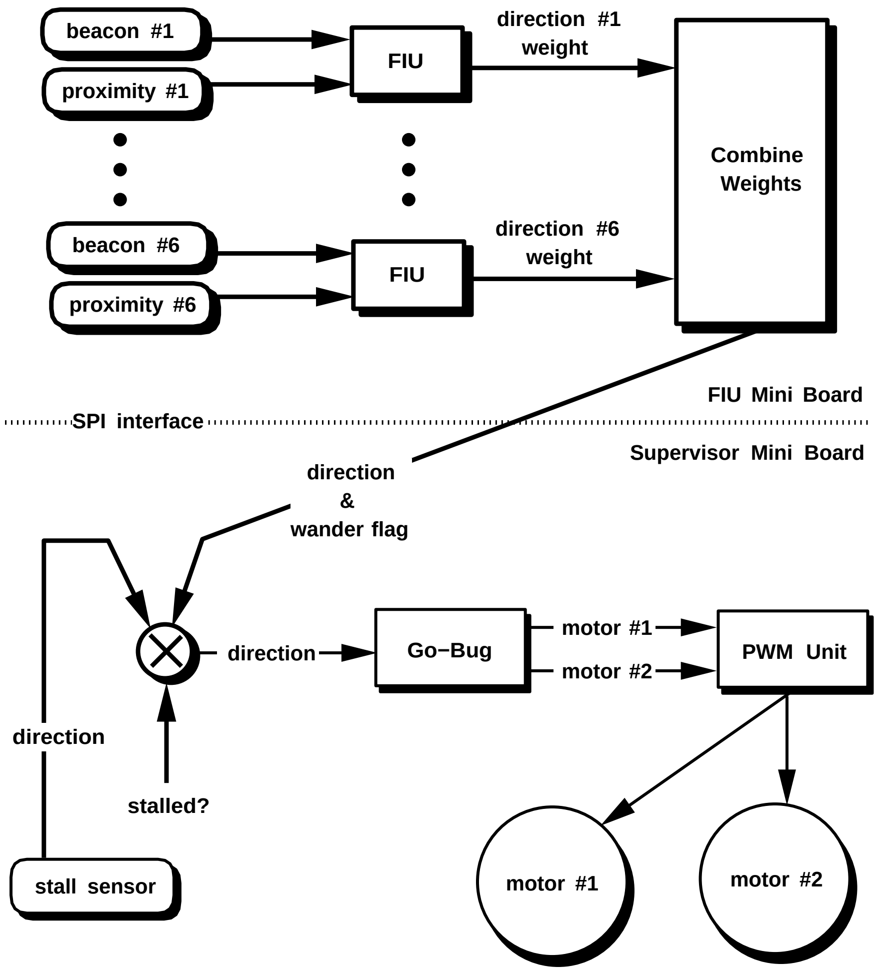

Figure 4: Data flow of the FuzzBug controller

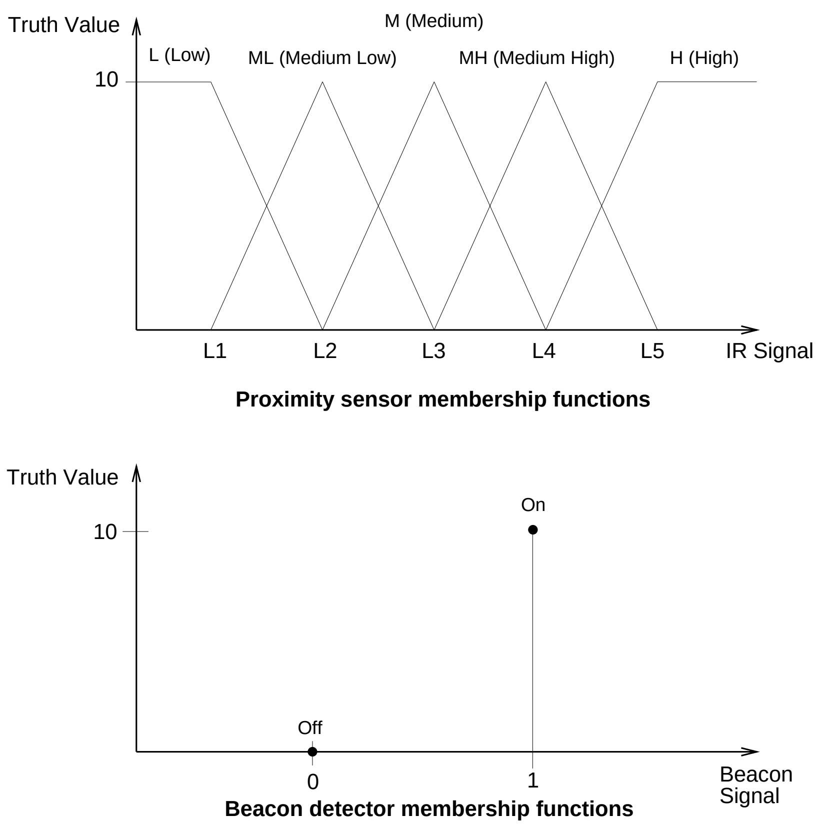

Figure 5: Input membership functions of the FIU. L1 – L5 were determined experimentally.

and MH = Medium High) represent varying degrees of blockage. The beacon sensor is a digital signal. Thus, its fuzzy subsets are rather crisp in nature. Off is a reading of 0 and indicates that the beacon was not detected. On is a reading of 1 and indicates that the beacon was detected.

The FIU takes as input a proximity sensor reading (0 to 255) and a beacon sensor reading (0 or 1). The first thing that is done is that the proximity sensor input is fuzzified. This is done by determining the grade, or truth, of each of the five fuzzy subsets (L, ML, M, MH, and H) of the proximity sensor domain. Due to certain limitations of the programming environment (no support for floating point numbers, for example), the grades of these subsets have a range between 0 and 10 (instead of between 0 and 1 as is normal). The output of this step is five truth values. For example, the following might result:

$$\begin{aligned}
\text{truth of L} &= 7 \\
\text{truth of ML} &= 3 \\
\text{truth of M} &= 0 \\
\text{truth of MH} &= 0 \\
\text{truth of H} &= 0
\end{aligned}$$

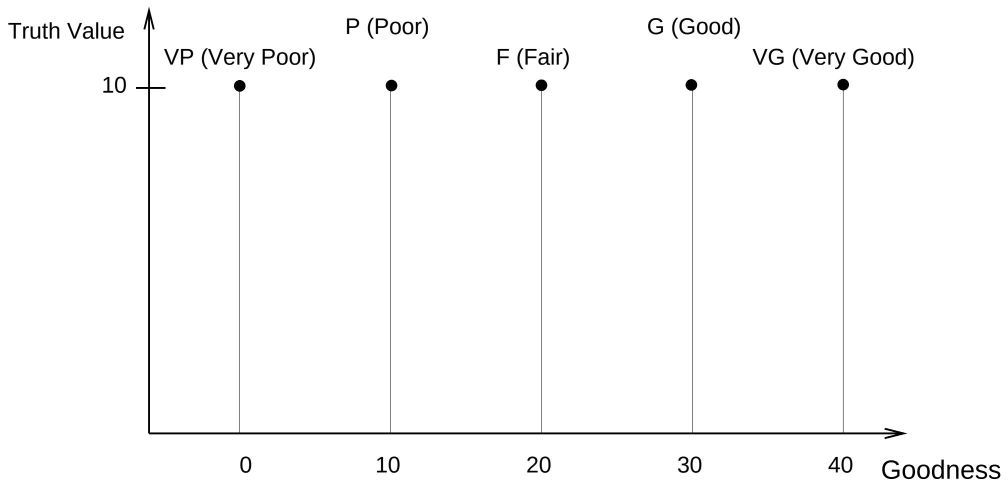

Figure 6: Direction (output) membership functions of the FIU

The next step is the fuzzification of the beacon sensor input. Although this sensor is digital in nature, it was treated as if it were possible to get values between 0 and 1. In this step, the grades, or truths, of each of the fuzzy subsets of the beacon sensor are determined. The output of this step is two truth values. For example, the following might result:

$$\begin{aligned} \text{truth of Off} &= 0 \\ \text{truth of On} &= 10 \end{aligned}$$

After the input data has been fuzzified, the fuzzy inference rules are executed. The output of the inference process is a measure of the goodness (or badness) of the direction to which the proximity sensor and beacon sensor belong. This measure is represented as a weight, with 0 indicating that the direction is Very Poor (VP), and a 40 indicating that the direction is Very Good (VG). The fuzzy subsets of this direction weight domain are shown in Figure 6.

The fuzzy inference rules follow:

1. if (proximity is H) then (direction is VP)
2. if (proximity is MH) and (beacon is Off) then (direction is VP)
3. if (proximity is MH) and (beacon is On) then (direction is P)
4. if (proximity is M) and (beacon is Off) then (direction is P)
5. if (proximity is M) and (beacon is On) then (direction is F)
6. if (proximity is ML) and (beacon is Off) then (direction is F)
7. if (proximity is ML) and (beacon is On) then (direction is G)

8. if (proximity is L) and (beacon is Off) then (direction is G)
9. if (proximity is L) and (beacon is On) then (direction is VG)

(This FIU uses the min function for AND and the max function for OR.) Notice that Rule 1 is designed to steer the FuzzBug away from a very close obstacle, no matter whether the beacon is detected in that direction or not. Thus, when an obstacle gets very close to the FuzzBug, the obstacle avoidance behavior takes precedence over the beacon homing behavior. This is desirable as the beacon cannot possibly be reached if an obstacle is hit, and damage may occur to the vehicle. The other rules are grouped in pairs as they correspond to the same proximity sensor reading. The only difference is whether or not the beacon is detected. Detecting the beacon has the effect of increasing the goodness of that direction. Rule 9 is important – when no obstacles are detected, the rule drives the FuzzBug in the direction of the beacon. This is because the only way the direction can be VG is to have a clear path with the beacon detected in that direction.

The output of the fuzzy inference rules is a grade, or truth value, for each of the five fuzzy subsets of the direction weight domain. For example, the following might result:

$$\begin{aligned}\text{truth of direction is VP} &= 0\\ \text{truth of direction is P} &= 2\\ \text{truth of direction is F} &= 6\\ \text{truth of direction is G} &= 4\\ \text{truth of direction is VG} &= 0\end{aligned}$$

The last step in the fuzzy inference process is to defuzzify these truth values. This must be done, as we only want one number to represent the goodness of each direction. In FIDE terminology, the TVFI defuzzification method is used. This method is simply a weighted average. For the example shown above, this would result in the following goodness measure for this direction:

$$weight = \frac{0 \times 0 + 2 \times 10 + 6 \times 20 + 4 \times 30 + 0 \times 40}{0 + 2 + 6 + 4 + 0} = 22$$

$$\begin{aligned}\text{where } 0 &= \text{weight of VP}\\ 10 &= \text{weight of P}\\ 20 &= \text{weight of F}\\ 30 &= \text{weight of G}\\ 40 &= \text{weight of VG}\end{aligned}$$

Thus, this direction would have a weight of 22, Since a weight of 0 is very poor (VP) and a weight of 40 is very good (VG), this direction is slightly above fair (F). This is easy to see by the truth values resulting from the fuzzy inference rules in the example.

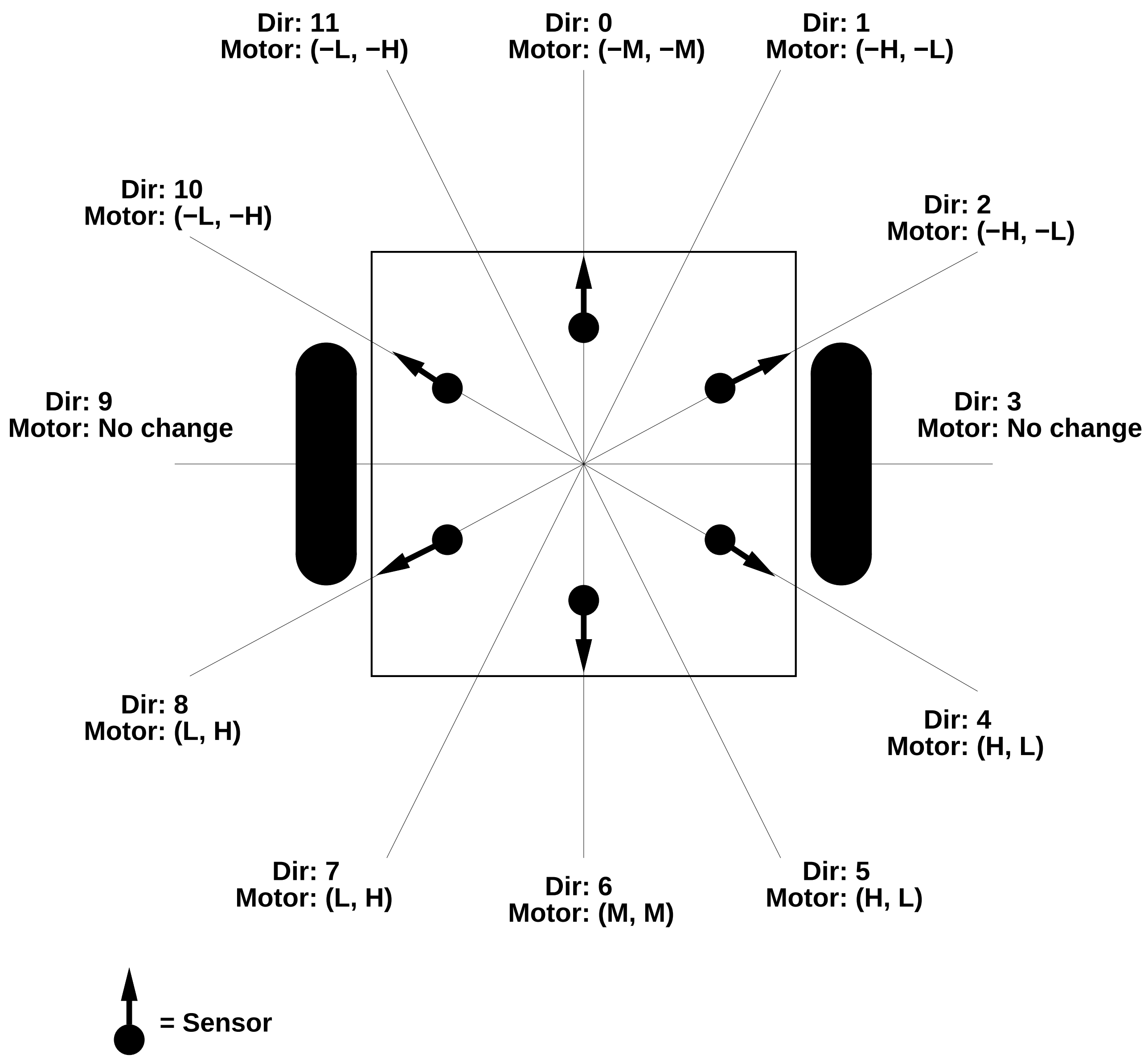

Figure 7: Sensor directions and corresponding motor speeds

This entire inference process just described is repeated for each of the six sensor directions. After this is done, we have six weights representing the goodness measures of the six sensor directions. We now need to combine these in order to choose the best of the six directions, or actually to choose a best direction even if it does not lie along one of the six sensor directions. In the FuzzBug we have twelve possible directions of motion. There is one direction for each of the sensor directions and one direction between each sensor direction and its neighboring sensor directions. This is shown in Figure 7.

The second level of the inference process performs a weighted average much like that done in the defuzzification step of the FIU. However, due to the circular nature of the directions and the problem of wrap-around, the algorithm is a bit more tricky. The output of this module is a direction from 0 to 11. In addition, a flag is included in this output to determine whether or not there actually is a preferred direction. A preferred direction does not exist if there are no obstacles detected and no beacon is detected. In this case, all directions have a weight of 30 and none are preferable to any others. In this situation the FuzzBug must choose a direction on its own. The wander flag lets the FuzzBug know when this situation is in effect.

The routine which performs the second level processing must take the array of 6 weights from the FIU and decide which direction to command the robot. The problem is that the array is circular. This means that a simple weighted average will not properly handle some cases. Therefore, the algorithm works as follows:

1. Calculate the maximum weight, and the number of directions having that weight.
2. Calculate the shortest path around the circle which visits all maximums.
3. Compute the average of the indices starting at the first element in that path, proceeding to the last.

The indices used to compute the average increase monotonically, and then a modulo 6 is taken to arrive at the final direction. For example, if the maximums are at sensors 0, 1 and 5, the average should be calculated as $(5+6+7)/3 = 6$. $6 \mod 6 = 0$, so the final direction should be 0. Actually, the sum is doubled before the division, and a modulo 12 is used. This is done so that we can get 12 distinct directions.

To do step 2 above, the following technique is used:

2.1. For each direction, place a 0 in the corresponding bit position of a 6 bit number if that direction does not contain a maximum. Place a 1 in that bit position if it does represent a maximum weight. For the example above, the bit pattern would be 110001.

2.2. Rotate the bit pattern five times, each time keeping track of the maximum value attained by the bit pattern and the number of shifts it took to get that maximum value.

2.3. After 5 shifts, the number of shifts required to find a maximum value of the bit pattern corresponds to the index to begin the average computation in step 3. For example, after 5 shifts, the bit pattern in step 2.1 will be 111000. This is the maximum value that is obtainable by rotating this bit pattern. Therefore, we should begin the weighted average computation at index 5.

B. *Supervisor Mini Board*

The supervisor Mini Board (supervisor) requests data from the FIU Mini Board at a rate of 20 Hz. This data consists of a direction (from 0 to 11) and a wander flag, as described in the preceding section. The supervisor consists of the following modules: 1) stall sensor module, 2) Go-Bug module, and 3) PWM motor control module.

The stall sensor module is called each control cycle to determine if the FuzzBug has stalled. A stall could occur if the FuzzBug has become entangled with an obstacle, hit a crack or other small obstacle, etc. If a stall is detected, the direction passed to the supervisor by the FIU is not used. Instead, the stall sensor module performs a pre-programmed maneuver designed to free the FuzzBug. Control returns to the supervisor after the completion of this maneuver.

If no stall is detected, the supervisor uses the direction passed to it by the FIU. To be used by the FuzzBug, this direction must be converted to a motor speed for each of the two motors. This is the job of the Go-Bug module. Go-Bug is a rather simple program that performs a table lookup to determine the motor speed for the two motors. Figure 7 shows how each direction is coverted to a motor speed for the left and right motors. In this figure, L stands for Low motor speed, M for Medium motor speed, and H for High motor speed.

The motor speeds resulting from Go-Bug's table lookup are sent to the PWM motor control module. This function uses Pulse Width Modulation (PWM) to control the two motors of the FuzzBug, and was included with the Mini Board as a library routine.

As mentioned in the preceding section, there may not actually be a preferred direction for the FuzzBug to follow. This occurs when no obstacles are detected and no beacon is detected. In this case, the FuzzBug is on its own and must choose its own direction, which it does by simply continuing in its current direction.

C. Communication between FIU and supervisor Mini Boards

Communication between the FIU Mini Board and the supervisor Mini Board takes place over the Serial Peripheral Interface (SPI) of the 68HC811 processor. The protocol for this interface requires that one board become the master and the other a slave. In the FuzzBug, the supervisor Mini Board becomes the master while the FIU Mini Board becomes the slave. The slave is at the mercy of the master, as only the master can initiate a data transfer.

The supervisor initiates a data transfer by writing into the SPDR register (a data register) of the SPI interface. At the start of every control loop, the supervisor reads and then writes to the SPDR register until a known bit pattern is received from the FIU Mini Board. Once this bit pattern is obtained, the supervisor retrieves the data from the FIU. It should be noted that, on the FIU Mini Board, the communication is handled by an interrupt service routine.

VI. Performance Evaluation

To evaluate the performance of the FuzzBug, several tests were run. We will first discuss the tests and the results qualitatively. Later, quantitative results will be presented to show the response under different environmental conditions.

A. Qualitative Results

The first step was to test the beacon-tracking performance, without obstacles, using a stationary beacon. In this situation, the FuzzBug responded almost instantaneously. The beacon was easily detected, and the FuzzBug responded by turning toward the beacon. This test was repeated, using different directions and varying the distance from the FuzzBug to the beacon. The FuzzBug could respond from most angles. Response was not as good from the side positions (i.e., 90 degrees and 270 degrees). This behavior can be explained by describing the conversion from commanded direction to motor control. The conversion method allows the FuzzBug to either go forward or backward, depending on which direction requires the lesser turn. When the beacon is directed at 90 degrees, the sensed direction sometimes "bounces" back and forth across the 90 degree line. This has the effect of making the FuzzBug oscillate back and forth as the direction changes. This behavior was corrected by forcing the FuzzBug to go straight instead. The rationale is that eventually, the beacon will be detected in another direction, causing the FuzzBug to turn correctly.

With varying distances, we found that the FuzzBug would sometimes get confused when the beacon was too close (one foot or less). In this situation, the IR LED beacon appears to be easily deflected, so that the beacon is detected in several directions. This would, of course, have the effect of confusing the navigation control. With larger distances (up to 12 feet or so), the beacon was detected accurately. The conclusion of these tests was that the navigation method performed well, within the limitations of the sensors.

Next, we tested the beacon-tracking performance, without obstacles, using a moving beacon. The results of this test were similar to those of the first test, using a stationary beacon. Within the limitations discussed above, the FuzzBug was able to track the moving beacon and respond by turning toward the target.

The next test was designed to evaluate the collision avoidance performance. The FuzzBug was tested in a field of obstacles, without a beacon target. The performance was generally good, within the limitations of the IR proximity sensors. Because the IR sensors are light-sensitive, shadows tend to be sensed as obstacles. Also, different obstacles reflect the IR beam differently, depending on the color and whether the surface is shiny or matte. In the tuning of the membership functions, we took a conservative approach, assuming that it was better to avoid all obstacles, including imagined ones. The performance of the FuzzBug reflects this decision. The FuzzBug will back up when it reaches a shadow. However, it generally can sense obstacles soon enough to avoid them. If it does happen to touch an obstacle because it did not sense it soon enough, it will at least back up when the obstacle is sensed.

Finally, the FuzzBug was tested in a field of obstacles, with a beacon target. Again, the performance was good, within the limitations described above. The beacon target was easily detected, whether stationary or moving. The FuzzBug was able to track the beacon, while at the same time avoid obstacles. Occasionally, the FuzzBug would get too close to an obstacle so that it would get "stuck". The stall sensor is used to rectify this condition.

Because the FuzzBug is purely a reactive control system and does not use any global path planning, it is possible for the FuzzBug to become stuck in a local minima. This reflects the limitations of all local path planning methods.

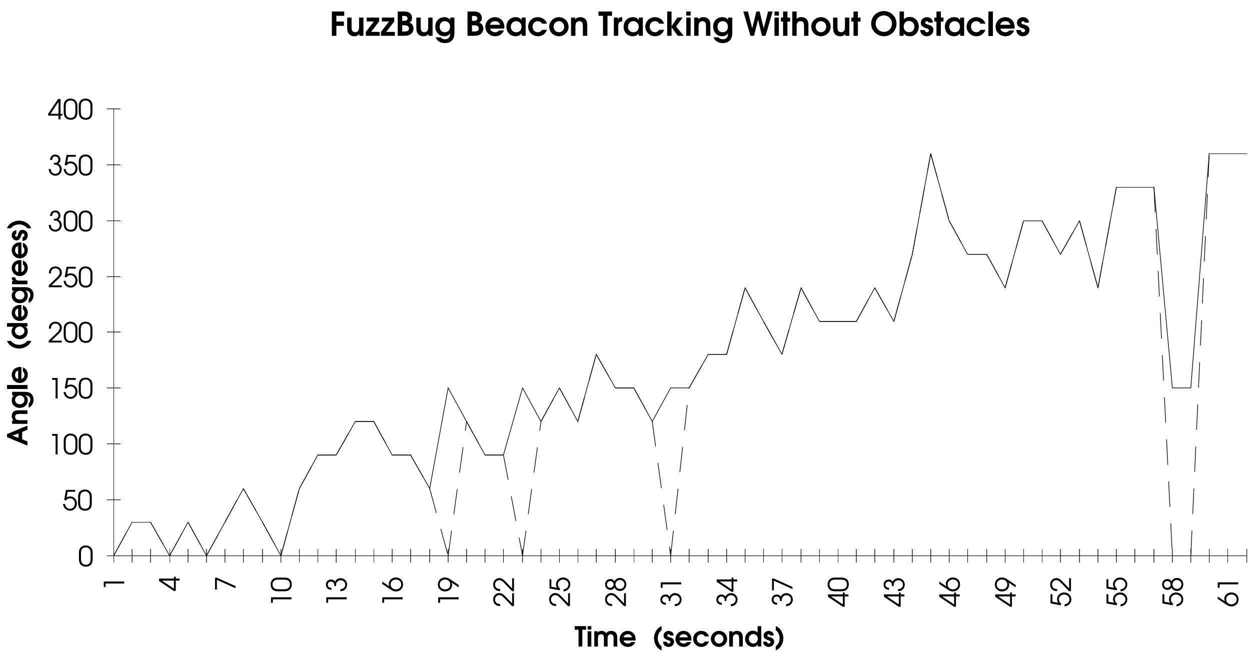

Figure 8: Results of defuzzification under normal lighting conditions and no obstacles. The dashed line indicates the direction in which the beacon was detected. The solid line indicates the resulting direction command.

B. *Quantitative Results*

To collect quantitative results of the FuzzBug's navigation system, data was collected with a stationary FuzzBug. The beacon target was then moved around the FuzzBug in a 360 degree circle. A test program was downloaded to the supervisor Mini Board, which simply displayed the results transferred from the FIU Mini Board. A log file was generated using Kermit. In this way, the commanded direction and beacon sensor information was captured so that it could be plotted.

Data was collected for four situations. Under normal lighting conditions, the beacon-tracking was measured both without any obstacles and with one obstacle at a known position. The tests were then repeated under a bright light condition. Figures 8, 9, 10, and 11 show the results.

Under normal lighting conditions and without obstacles, as shown in Figure 8, the FuzzBug was able to track the beacon quite well. The four "blips" represent those times when the FuzzBug did not sense the beacon at all. In all other cases, the commanded direction was the same as the sensed direction of the beacon. This test may appear somewhat trivial; however, with noisy and inconsistent sensors, the beacon is often detected by more than one sensor and not necessarily adjacent sensors. As a result, the pattern of the final weights can sometimes be confusing. As shown in the graph, the defuzzification algorithm used to combine the weights worked quite well.

In the second test, shown in Figure 9, one obstacle was placed at 120 degrees. In this case, when no beacon is detected, the commanded direction would be 300 degrees (i.e., 180 degrees away from the obstacle). When a beacon is detected, the commanded direction should be essentially the same as the beacon's direction until the beacon gets too close to the obstacle. Then, the commanded direction should turn the FuzzBug away from the obstacle, but hopefully still in the same general

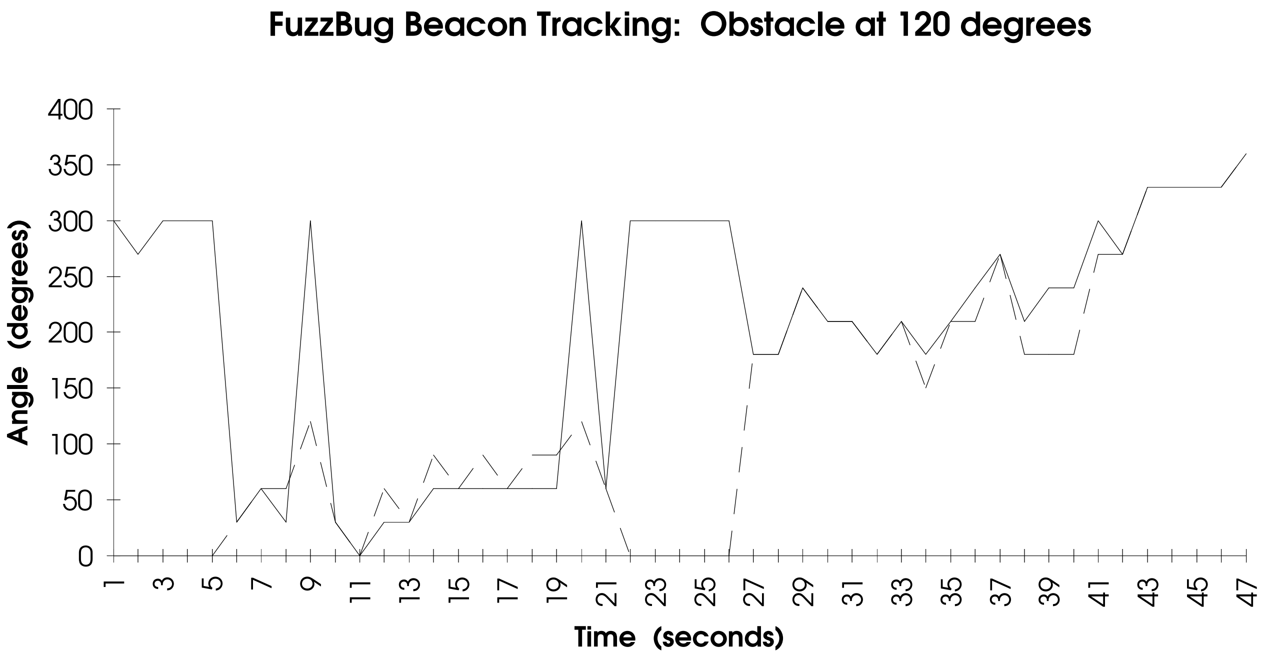

Figure 9: Results of defuzzification under normal lighting conditions and one obstacle at 120 degrees. The dashed line indicates the direction in which the beacon was detected. The solid line indicates the resulting direction command.

direction as the beacon. The second graph reflects these results. Initially, the beacon is not turned on, and the commanded direction is 300 degrees. Later, when the beacon is hidden by the obstacle, the commanded direction is again 300 degrees. As the beacon approaches the direction of the obstacle, the commanded direction turns the FuzzBug slightly away from the obstacle. At two points, when the beacon is detected right at 120 degrees, the obstacle's presence cancels the beacon direction, and the commanded direction is set to 300 degrees. This behavior represents a tradeoff in design goals. The rules and membership functions could be designed to turn the FuzzBug slightly away from the obstacle. However, in real-world operation, a back up is actually more useful.

The last two graphs show the same tests under a bright light condition. This is intended to test the adaptability of the system, especially since the IR sensors are known to be light-sensitive. Without any obstacles, the beacon-tracking performance was good, as shown in Figure 10. Again, there are a few blips where the beacon was not seen at all. The performance was not as good when an obstacle was added, as shown in Figure 11. In this case, the IR sensors did not consistently sense the obstacle, and the commanded direction was sometimes 120 degrees. Given the operation of the IR sensors, this behavior was expected. The performance could be improved by adding photo sensors to sense the ambient level of light. This information could then be used to change the membership functions of the IR sensors on-line. This would correspond to changing the values of points L1 through L5 in Figure 5. We did not implement this feature because of time and memory constraints.

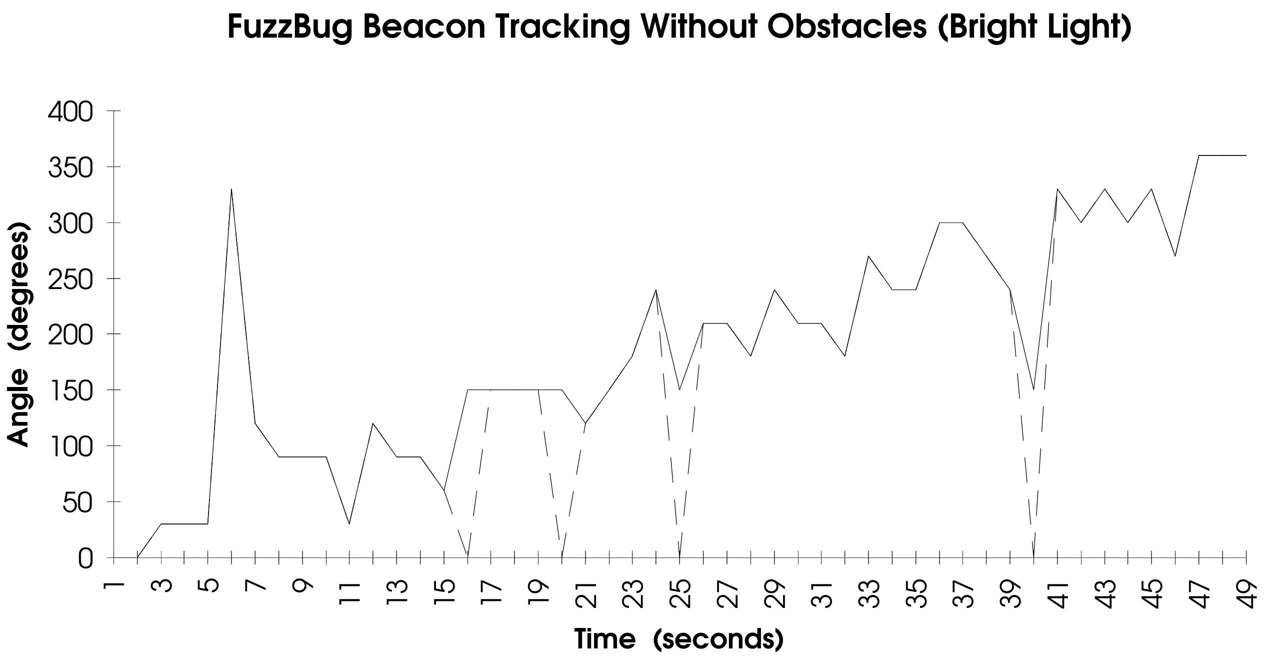

Figure 10: Results of defuzzification under bright lighting conditions and no obstacles. The dashed line indicates the direction in which the beacon was detected. The solid line indicates the resulting direction command.

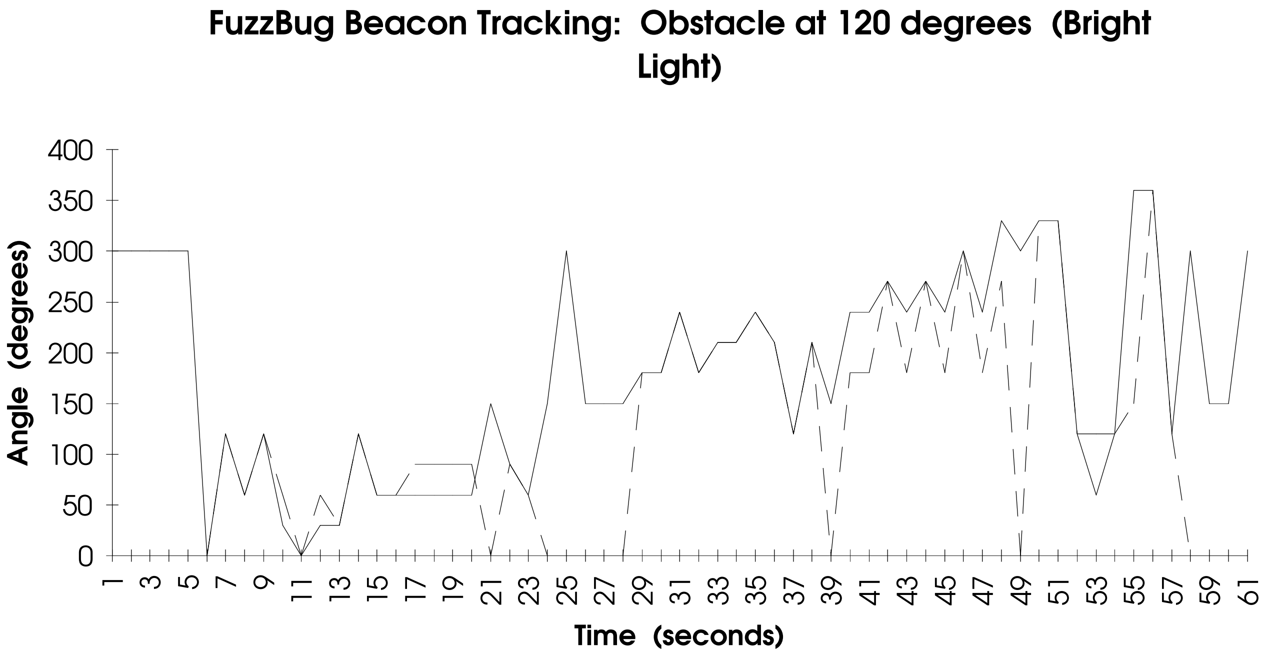

Figure 11: Results of defuzzification under bright lighting conditions and one obstacle at 120 degrees. The dashed line indicates the direction in which the beacon was detected. The solid line indicates the resulting direction command.

VII. Comparison to Non-Fuzzy Systems

Reactive control of mobile robots has also been implemented using a subsumptive behavior approach [3, 4]. As applied to the FuzzBug, this would involve creating separate behaviors for the beacon-tracking and the obstacle avoidance functions. The problem with this approach is that one behavior becomes dominant over the other behavior, and the two functions are not fused very well. That is, if the FuzzBug encountered an obstacle while tracking the beacon, it would not necessarily know the best way to turn, while still tracking the beacon. The advantage to using fuzzy logic is that the two functions can be fused in a very natural way, so that a reasonable decision can be made to serve both functions.

Another advantage of using fuzzy logic applies to the noisy IR sensors. Using a discrete method, the IR sensors would either show the complete presence of an obstacle or no obstacle at all. The boundary would be a single threshold value. Unfortunately, with light-sensitive sensors, this threshold would change depending on the ambient level of light. This condition exists in the fuzzy implementation as well, but using fuzzy logic allows the boundary to be gradual instead of a sharp line. This has the effect of smoothing out the effects of the variations. If a discrete method were used, the performance would probably be even more sensitive to the lighting conditions.

VIII. Conclusion

In general, the FuzzBug demonstrated that inexpensive components could be used to implement a mobile robot. Fuzzy logic was shown to provide a reasonable navigation system, even when using noisy sensors. A big advantage of using fuzzy logic is that competing functions can be combined in a reasonable way so that an optimal solution can be reached.

The two-level fuzzy logic controller proved to be a efficient method for combining multiple sensors with a small amount of memory. The alternative would be to write one set of rules that referenced all the sensors. This approach would have been much more complex and would have used considerably more memory.

Although the FuzzBug works well, a number of improvements could be made. For more adaptable performance, readings from photo sensors could be used to adjust the membership functions of the IR sensors. This could be done on-line, to provide adaptability in real-time. Given more time and more memory, we would have implemented this feature.

Another improvement would be to incorporate the stall sensor feedback into the fuzzy control system. Under the existing architecture, the stall sensor feedback is used to provide control independent of the other sensors. In certain situations, it would be possible for the FuzzBug to be directed into an obstacle while performing a back-up maneuver generated by a stall condition.

Other improvements could also be made if more sophisticated sensors were used. Certainly, obstacle avoidance would be easier if we had used sensors which provided better range information. However, part of the challenge was to use inexpensive sensors and test how well the system performed. Considering the limited information available from the sensors, the FuzzBug performed quite well.

IX. Acknowledgement

We would like to thank Dr. Richard Volz for his support and for permitting our use of the facilities of the Robotics Lab. We also thank Charath Ram and Soon-Lin Yeap for their assistance.

References

[1] K.-T. Song and J.-C. Tai, “Fuzzy navigation of a mobile robot,” In *Proceedings of the 1992 IEEE/RSJ International Conference on International Conference on Intelligent Robots and Systems*, pp. 621–627, July 1992.

[2] J. Yen and N. Pfluger, “Path planning and execution using fuzzy logic,” In *Proceedings of the AIAA Conference on Guidance, Navigation, and Control*, volume 3, pp. 1691 – 1698, New Orleans, LA, August 1991.

[3] R. A. Brooks, “A robot that walks: Emergent behaviors from a carefully evolved network,” In *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, N. Badler et al., editors, chapter 4, pp. 99–110, Morgan Kaufmann, 1991.

[4] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey, “Plan guided reaction,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 6, pp. 1370–1382, November 1990.

A. Hardware Schematics and Diagrams

B. Code Listings