Int. J. Engng Ed. Vol. 21, No. 1, pp. 178–186, 2005
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2005 TEMPUS Publications.

Implementing Fuzzy Logic for Machine Intelligence: A Case Study*

M. D. HURLEY, W. L. XU and GLEN BRIGHT
Institute of Technology and Engineering, College of Sciences, Massey University, Palmesrton North, New Zealand. E-mail: W.L.Xu@Massey.ac.nz

Intelligent machines are machines with microcontroller(s), actuators and sensors embedded and of reprogrammable intelligence. The Mechatronics course at Massey University, New Zealand, teaches students how to design intelligent machines in an integrated manner. The success of the teachingllearning would be achieved in a way that the students are able to apply what they have learnt from various courses to the design of an intelligent machine. To this end, each student is required to do a design project. This paper presents a sample project that deals with real-time implementation of fuzzy logic for machine intelligence on microcomputer. The objectives identified for the project were to modify Rug-Warrior Pro, an autonomous mobile robot platform, to have a long-ranging capability and to implement obstacle avoidance in an unmapped and changing environment as the machine intelligence. The hardware interfacing and software drivers of the sensors are given, and the techniques for coding the membership functions and defuzzification operation of fuzzy logic are discussed. The machine behaviours are formulated by an If–Then rule base that mimics human heuristic, and the resultant program offers an excellent alternative to more common vector-based navigation methods with a fraction of the processing requirements resulting in a fast-responding, reliable application.

INTRODUCTION

FUZZY LOGIC HAS found many applications in machine intelligences and behaviours based controls, and its implementations can be either on PCs or microcontrollers [1]. The student project presented in this paper was to develop a fuzzy logic approach to obstacle avoidance for an educational mobile robot and to implement it on the embedded microcomputer. The mobile robot used in the project is Rug-Warrior Pro, which was developed by the Massachusetts Institute of Technology (MIT) for use in robotics courses [2]. It is a small autonomous robot designed for educators, researchers, and hobbyists and works in an unmapped and changing environment. The Rug-Warrior Pro is a microprocessor-based mobile robot kit that contains the complete processing, memory, and sensor circuitry (Brains), as well as the motors, wheels, chassis and custom body parts (Brawn).

The Rug-Warrior Pro may be programmed from either a Mac or an IBM PC using Interactive C. Interactive C is a compilation environment for many Motorola 6811-based robots and embedded systems. The code is downloaded from Interactive C to the Rug-Warrior Pro via the serial port. Library routines and self-test code are provided for all the robot's standard sensors and actuators, such as motor routines and printing to the LCD. The Rug-Warrior Pro can be controlled directly from the keyboard by way of Interactive C's command line or it is able to operate autonomously under the control of its on-board microcontroller. The 'brains' of the Rug-Warrior Pro is the Motorola MC68HC811E2 microcontroller. This uses the Motorola MC68HC11 microprocessor with extended memory. It contains 256 bytes of RAM, and 2048 bytes of EEPROM.A time-sharing operating system allows the execution of parallel processes. For example, one process may be used for motor control, while another independent process is used for reading and processing information from the sensors [2, 8].

The machine behaviours are formulated by a set of If–Then rules that mimic human heuristic and are coordinated in fuzzy logic. To have a long-ranging capability for the purpose of obstacle avoidance, the Rug-Warrior Pro was modified to include three sets of ultrasonic sensors. The hardware interfacing and software drivers of the sensors are given, and the techniques for coding the fuzzy logic operation are discussed in the paper. The learning outcomes achieved in the project for the course *Mechatronics* are sensor and signal processing, motor drive and control, microcontroller interfacing and software driver, machine intelligence and programming, and integration of mechatronic components. The project presented as a case study promotes students' can-do attitude in implementing what they have learned in advanced topics including machine intelligence.

* Accepted 20 May 2004.

Table 1. Sensor interfacing configuration

Sensor Set	Enable Pin	Output
Left	PD2	PE7
Centre	PA3	PE6
Right	PD3	PE5

OBSTACLE SENSORS AND THEIR INTERFACING AND SOFTWARE DRIVERS

Ultrasonic sensors

For obstacle detection operation, an ultrasonic ranging system was chosen. The sensor system used was a Velleman Parking Radar kit available from Dick Smith Electronics stores. The output of the sensor kit is either high or low; no ranging data is provided from the kit [7]. The ultrasonic detection module works on a very simple principle. It transmits a 40 kHz signal from an ultrasonic transducer and, with the receiving transducer, the echo from the transmitter is monitored at a sampling frequency of 26 Hz. When an echo is received, the resultant output from the receiver is then conditioned before being used to drive a piezo buzzer. The resultant output to the piezo buzzer is an active low signal on the negative pin (i.e. when an obstacle is detected, the signal is low). For the application of obstacle avoidance, the piezo buzzer was removed and the negative pin was connected to the Rug-Warrior's microcontroller.

Three sets of ultrasonic sensors (and so three Velleman Parking Radar kits) were required, one to detect objects on the left, one set for the centre and one set for the right. This meant that three digital inputs and three digital outputs were required. The digital inputs were available in the form of PE5, PE6 and PE7 (Port E, lines 5–7). These lines were used to receive the output (or echo line) from the sensors [2, 3]. In order to get the required three digital outputs, the two infrared emitters and the piezo buzzer were removed from the microcontroller. This left the digital output lines PD2, PD3 and PA3 available for the sensor enable lines.

Bump Switches

The Rug-Warrior Pro comes standard with three bump switches; one on the front left, one on the front right and one at the rear. Each bump switch has three pins, pin 3 being ground, pin 2 is V_{cc} and pin 1 is the output. When the switch is open, the output is ground, when closed the output is V_{cc}. The bump switches are connected to a 10-pin connector that is hard-wired on the microcontroller board. The analogue output for the bump switches is received on the PE3 line of the microcontroller [3]. When one or more of the bump switches are triggered, a binary value is returned to the microcontroller (via PE3). The return values are shown in Table 2.

For this application, only the two front bump switches were being used; the output line for the rear bump switch was hard-wired to ground, therefore only analogue outputs of 0–3 could be received from the bump switches.

FUZZY LOGIC FOR OBSTACLE AVOIDANCE

Why fuzzy logic?

For the obstacle avoidance algorithm, a fuzzy logic system was chosen. There are two main reasons why a fuzzy logic system was chosen: the first is due to the ultrasonic sensors that are used, and the second is because of processing requirements. Ultrasonic sensors in a distance ranging system such as the one used in this application are notorious for providing imprecise data, and using a fuzzy logic system aids in ensuring that this lack of precision has minimal effects on the overall functionality of the program. One of the main considerations for the implementation of this application was the processing speed of the microcontroller and thus the reaction time of the robot. With this in mind, it was necessary to minimise the number of instructions and calculations required.

Many obstacle avoidance techniques that do not use a fuzzy logic system work by dividing the area surrounding the robot into a grid pattern and then calculating the probability of an obstacle being present in each square of the grid based on the sensor data. These methods require a vast number of calculations and so significantly increase the response time of the robot. Fuzzy logic requires much fewer calculations and so is more suited for this application. For this application of fuzzy logic for machine intelligence, there were three basic steps to developing the program:

Table 2. Returned values for bump switches

Rear (Bit 2)	Left (Bit 1)	Right (Bit 0)	Returned Value	Analogue Value
0	0	0	000	0
0	0	1	001	1
0	1	0	010	2
0	1	1	011	3
1	0	0	100	4
1	0	1	101	5
1	1	0	110	6
1	1	1	111	7

- converting the obstacle detection algorithm to get distance-ranging data;
- developing a fuzzy logic algorithm based on the ranging data; and
- combining the ranging system with the fuzzy logic algorithm and the bump switches to get an overall obstacle avoidance program.

Distance ranging

The distance ranging algorithm is very simple and is conducted separately for each set of sensors. The enable bit is set to allow the signal to be set by the transmitter. For the left set of sensors, this is PD3, and Port D is located at address 0x1008 and 0b00001000 indicated bit 3 at that address [4]. That is, *bit_set (0x1008, 0b00001000).*

The internal clock on the microcontroller is then checked and the transmit time is stored as a variable. For the left set, it is *left_start*. The internal clock, also called the E-clock, is located at address 0x100E. Thus, *left_start = peekword (0x100E).*

The output or echo bit is then checked. If an echo has been received, the E-clock is again checked and the signal's time-of-flight is calculated. This time-of-flight is simply the time the echo is received minus the transmit time. The result is the number of machine cycles that have elapsed. Once the time-of-flight has been calculated, this is then converted into a distance by taking into account the speed of sound and the clock frequency. However, as the time-of-flight (or *ToF*) is the time taken for the signal to travel to the object and return again, this time-of-flight must be halved.

ToF = Number of Clock Cycles x Clock Period
= No. of cycles x 0.5 us
Distance = (ToF / 2) x Speed of Sound
= (ToF / 2) x 342 m/s

Both these calculations can be combined, giving the result as: *Distance = ToF x 0.0000855*, where the distance is in metres.

If no echo is detected, the result is that the distance is set to the maximum, which for this application is one metre.

```
PE7 = peek (0x100A) & 0b10000000;
if (PE7 = = 0)
            {         left_time = peekword (0x100E)
                      - left_start;
left_dist = (float) left_time * 0.0000855;
}
else
left_dist = 1.0;
```

Once the object distance has been calculated, the enable bit for the transmit signal is cleared, turning off the transmit signal. The operation is then returned to the main program returning the obstacle distance. That is:

```
bit_clear (0x1008, 0b00001000);
return (left_dist);
```

Membership functions

For a fuzzy logic operation, two main parts are required. The first is the membership functions

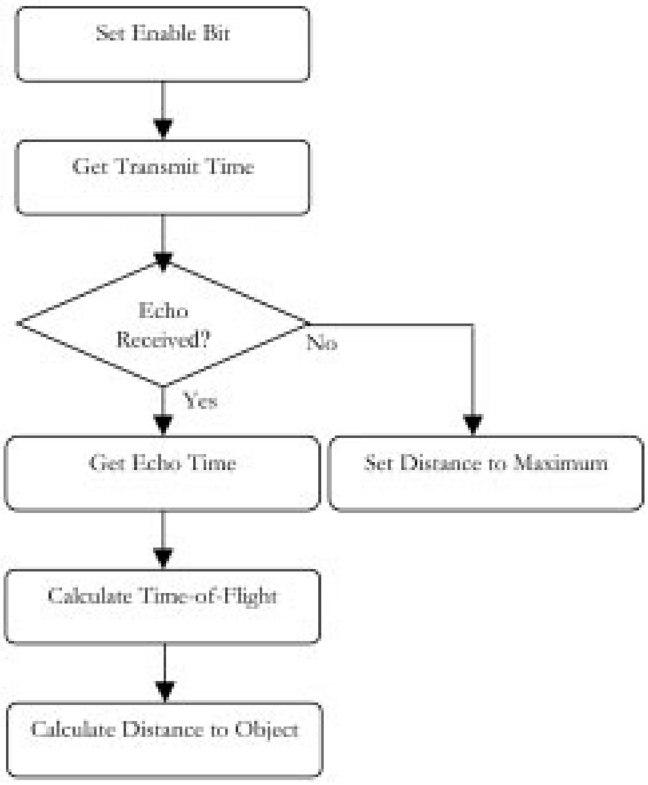

Fig. 1. Distance ranging algorithm.

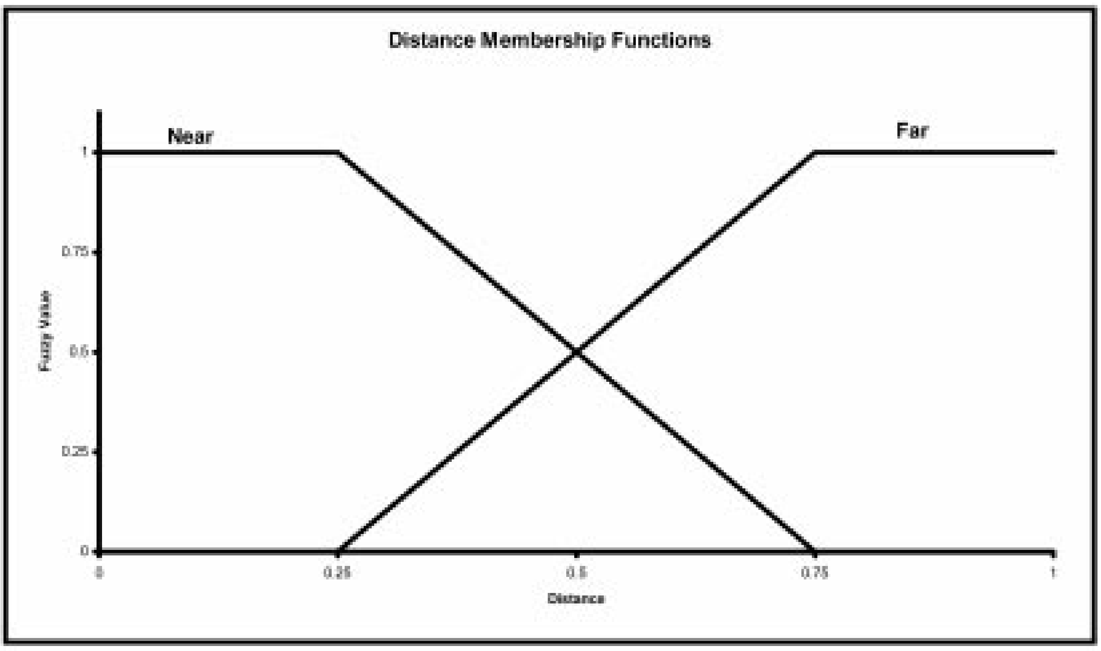

Fig. 2. Distance membership functions.

and, the other part that a fuzzy logic operation requires is the rule base, which defines the required outputs for any given combination of inputs [5, 6]. The membership functions are used for converting the discreet data from the inputs, in this case the ranging data, into a fuzzy value between zero and one, or converting the fuzzy output values into discrete values for output. For this application there were three sets of membership functions required, one for the inputs (distance) and two for the outputs (i.e., translational and rotational speed).

The ranging data (inputs) use the distance membership functions. They are used to convert the ranging data into near and far components. For distances less than 0.25 m (25 cm), near = 1 and far = 0. For distances greater than 0.75 m (75 cm), near = 0 and far = 1. For distances greater than 0.25 m and less than 0.75 m, the result is a combination of near and far [6].

In order to convert the ranging data into fuzzy values, these membership functions need to be converted into calculations that can be implemented using Interactive C. Below is the code that is used to do this.

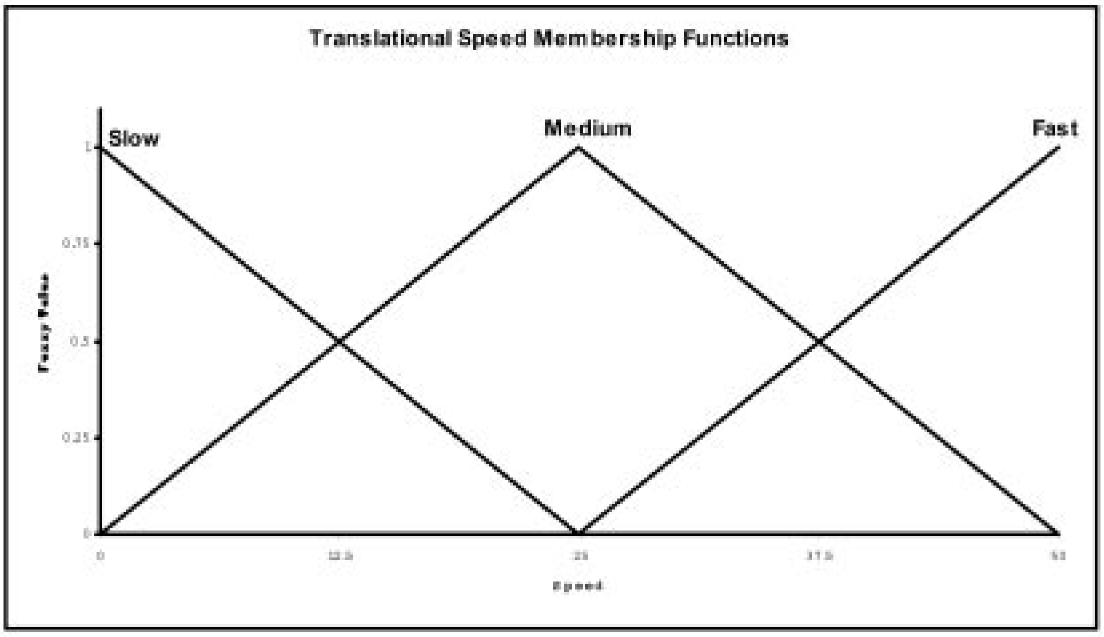

Fig. 3. Translational speed membership functions.

```
float NEAR (float dist_near)
{
        if (dist_near < = 0.25)
          return (1.0);
        else if (dist_near > = 0.75)
          return (0.0);
        else
          return ( (–2.0 * dist_near) + 1.5);
}
float FAR (float dist_far)
{
        if (dist_far < = 0.25)
          return (0.0);
        else if (dist_far > = 0.75)
          return (1.0);
        else
          return ( (2.0 * dist_far)—0.5);
}
```

There are two required outputs for the fuzzy logic system, the translational speed and the rotational speed. These two outputs are the values that are used for the robot's drive function. The translational speed is divided into three parts: slow, medium and fast, as shown in Fig. 3, with the maximum speed being 50 for this application. This is reflected as a percentage of the robot's maximum speed. The top speed is not used in this application, because slowing the robot down gives a better response as it allows more time for the robot to react as the environment changes. For the output variable, a fuzzy value for each parameter (i.e. slow, medium and fast) is received. These fuzzy values are combined to give the required output.

In order to convert the fuzzy values into a discrete speed, only one simple calculation is required. The fuzzy value for each parameter is multiplied by the speed at the maximum point of the membership function (refer to Fig. 3), then the three are added together. That is, the fuzzy value for slow is multiplied by 0.0, the medium fuzzy by 25.0 and the fast fuzzy value by 50.0. By adding these values together, the required speed can be found.

```
float DEFUZZIFY_SPEED (void)
{
        float speed;
        speed = (max_SLOW * 0.0)
          + (max_MED * 25.0)
          + (max_FAST * 50.0);
        return (speed);
}
```

Calculating the rotational speed output follows the same principle as the translational speed; however, the rotational speed has five parts: turn right big (TRB), turn right small (TRS), turn zero (TZ), turn left small (TLS), and turn left big (TLB). The outputs for rotational speed can be in the range of -50 to 50, where -50 is TLB and 50 is TRB (refer to Fig. 4).

Converting the rotational speed calculations to code for Interactive C uses the following function:

```
float DEFUZZIFY_SA (void)
{
float SA;
SA = (max_TLB*-
50.0) + (max_TLS*25.0) + (max_TZ*0.0)
  + (max_TRS*25.0) + (max_TRB*50.0);
return (SA);
}
```

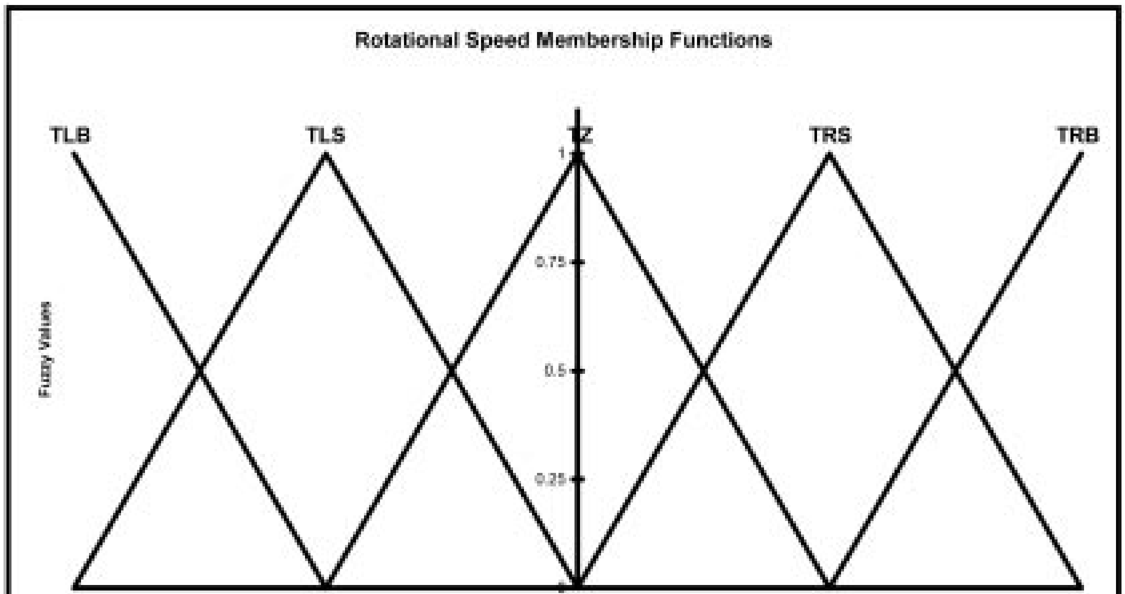

Fig. 4. Rotational speed membership functions.

Table 3. Rule base for robotic behaviours

Rule	Left	Centre	Right	Speed	Rotation
A	Near	Near	Near	Slow	TRB
B	Near	Near	Far	Medium	TRS
C	Near	Far	Near	Slow	TZ
D	Near	Far	Far	Fast	TRS
E	Far	Near	Near	Medium	TLS
F	Far	Near	Far	Slow	TRB
G	Far	Far	Near	Fast	TLS
H	Far	Far	Far	Fast	TZ

Fuzzy rule base

For a fuzzy logic system, the rule base defines the required outputs for any given combination of inputs. The variables for each input or output are defined by the membership functions. For the distance inputs, they are either near or far; for translation speed output, they are slow, medium or fast and for the rotational speed output they are turn right big (TRB) turn right small (TRS), turn zero (TZ), turn left small (TLS) or turn left big (TLB) [6]. For example, if an object is near to the left, centre and right sensors, then rule A applies (refer to Table 3). This would make the translation speed slow and the rotation turn right big (or TRB). In converting the rule base, the rules are expressed as a simple If–Then rule; for example, in the case of rule A, this is converted into an If–Then rule as:

IF (left = Near) & (centre = Near)
& (right = Near) THEN (speed = slow)
& (rotation = TRB)

Fuzzy logic operation

For calculating the fuzzy values for the outputs, it is necessary to combine the input fuzzy values with the rule base. The fuzzy value for each rule output is the smallest fuzzy value from the rule inputs. For example, take rule A:

IF (left = Near) & (centre = Near)
& (right = Near) THEN (speed = slow)
& (rotation = TRB)

To find the fuzzy values for the output, it is necessary to find the smallest input value. Taking the near component for the three inputs (left, centre and right), a calculation is done to find the minimum value. The Interactive C code is as follows:

```
float MIN (float min1, float min2, float min3)
{
        float min_val;
        min_val = min1;
        if (min2 < min_val)
                min_val = min2;
        if (min3 < min_val)
                min_val = min3;
        return (min_val);
}
```

In this case, the outputs are *SLOW* and *TRB*. The fuzzy value for the output is the smallest of the input values, so *SLOW* and *TRB* equal the result from the minimum calculation. By conducting this for each rule, fuzzy values for all the output parameters can be derived. Looking at the rule base (Table 3), it is obvious that there is going to be more than one value for each parameter. In this application there are either two or three values for each. This means it is necessary to decide what the final output is going to be. Using a maximum calculation does this according to the fuzzy inference [5]. The final value is the maximum value from the values calculated above. For example, rules A, C and F will all give a value for the output *SLOW*; the final value for this output is the largest of these three values. This is given in the Interactive C code procedure below:

```
float MAX (float max1, float max2, float max3)
{
        float max_val;
        max_val = max1;
        if (max2 > max_val)
                max_val = max2;
        if (max3 > max_val)
                max_val = max3;
        return (max_val);
}
```

Once this operation is complete, fuzzy values will have been found for each of the output parameters, which are subsequently defuzzified to get discrete values that may be used to drive the robot.

Overall machine intelligence

The overall functionality of the program is a combination of the bump switches, the ultrasonic distance ranging and the fuzzy logic operation. To begin, the status of the bump switches are checked. If one is closed, the robot will reverse and turn away from the object by 90; if both are closed, the robot will reverse and turn 180. When neither of the bump switches is closed (the normal case), the program receives ranging data from the sensors then uses this to conduct the fuzzy logic operations. The outputs from the fuzzy logic operation are a translational speed and a rotational speed, and these are used to drive the robot using default the library function: *drive (trans_vel, rot_vel).* Figure 5 shows the flowchart of the fuzzy logic obstacle avoidance algorithm, and Fig. 6 shows the

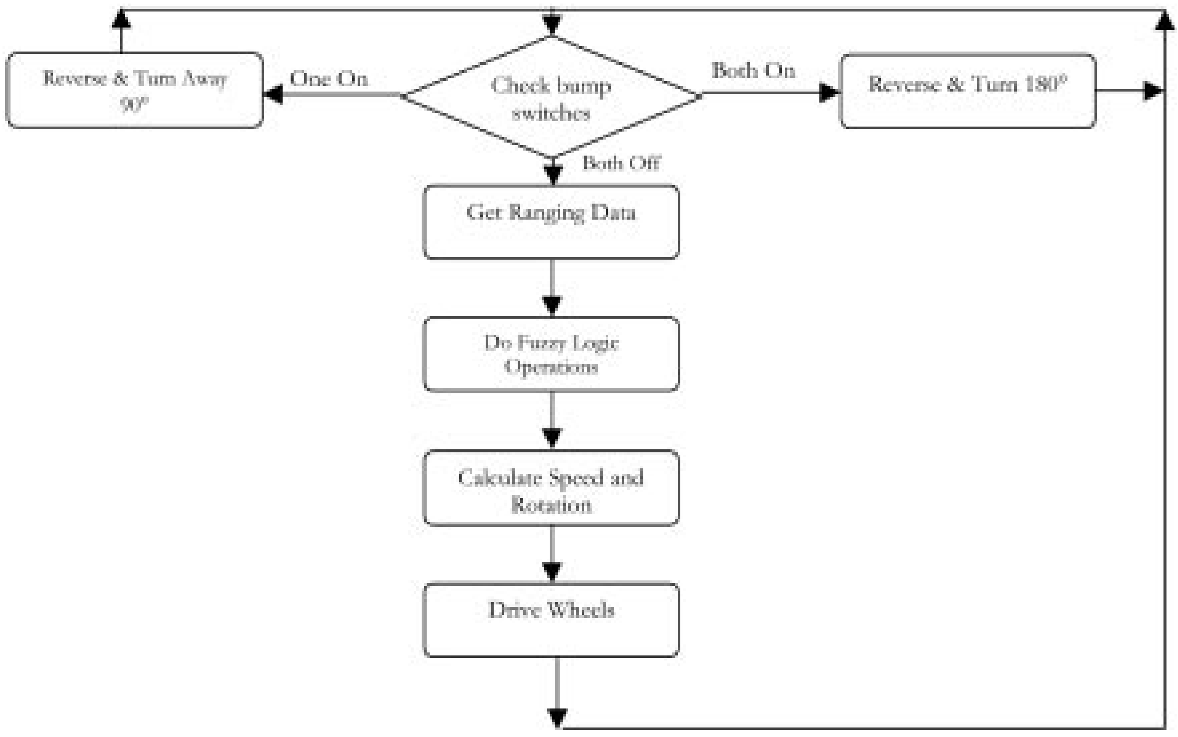

Fig. 5. Algorithm for fuzzy logic obstacle avoidance.

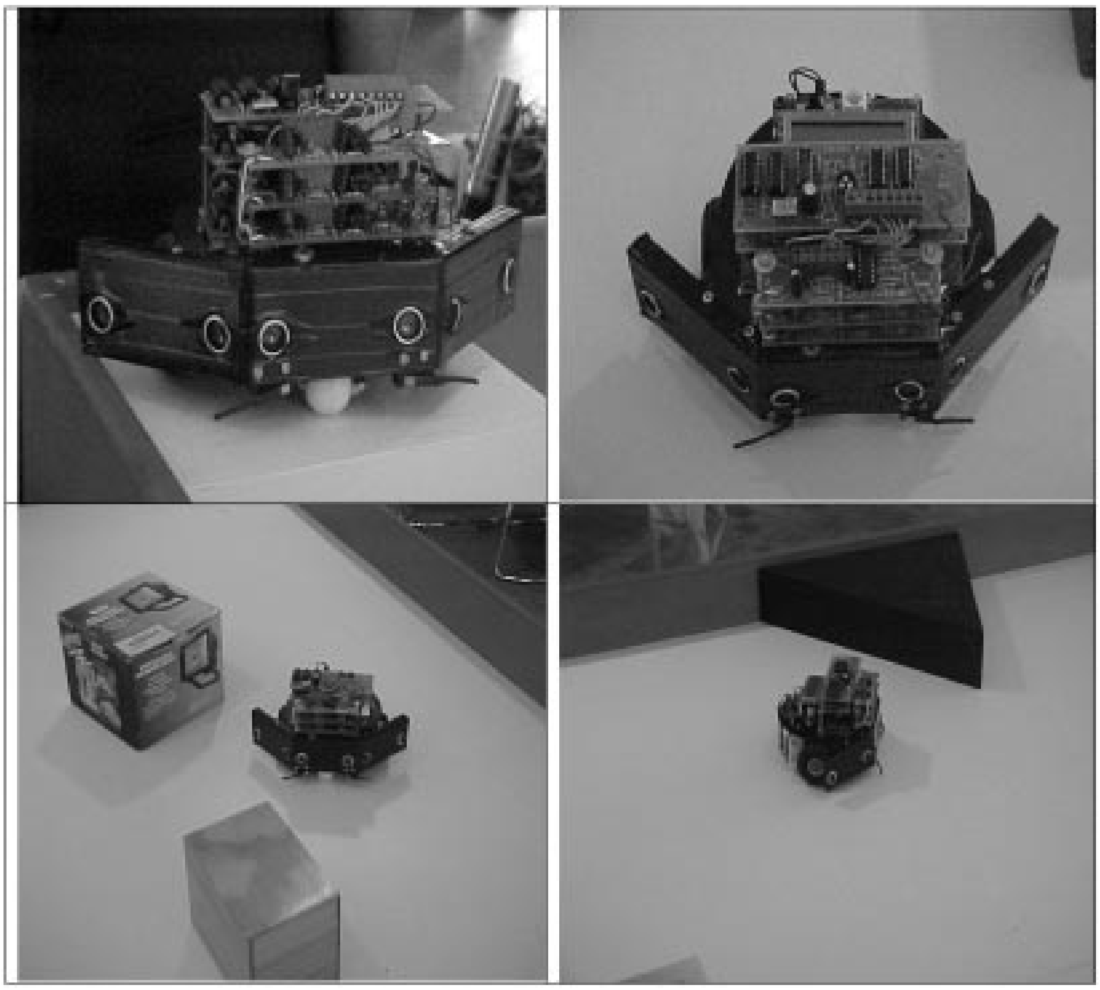

Fig. 6. The running robot with embedded fuzzy logic intelligence.

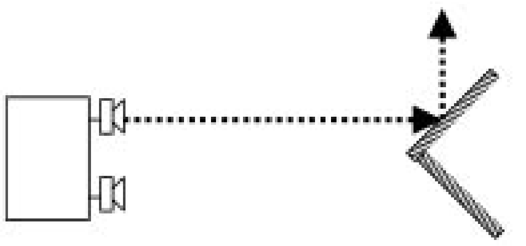
Fig. 7. Operation limitation on external corners.

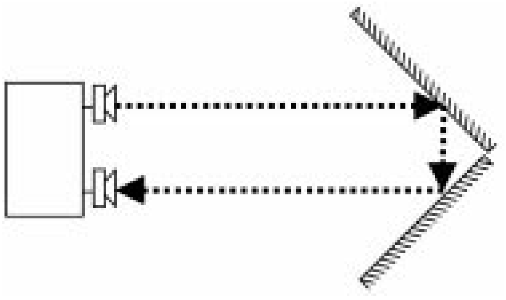
Fig. 8. Operation limitation on internal corners.

final robot and its negotiation of objects on a platform.

PROBLEMS AND LIMITATIONS

There are three main limitations for the overall operation of this application. These are due to the sensors used. The first is external corner deflection (Fig. 7), which occurs when the robot approaches an external corner. Instead of the transmitted signal reflecting back towards the receiver, it is deflected away, making it appear to the robot that no obstacle is present. For this application, the use of the bump switches is the only way that this problem may be overcome. When the robot makes contact with the obstacle, one or both of the bump switches will be triggered, causing the robot to reverse and turn away from the obstacle.

The second limitation of this application is caused when the robot approaches internal corners (Fig. 8). The transmitted signal reflects around the inside of the corner before returning to the receiver, which causes the signal to travel further and so causes ranging errors. Using a fuzzy logic system, these errors are minimised to a point where, for this application, they have little or no noticeable effect. However, for larger applications, they must be taken into account.

The final major limitation with this application results from the sensor kits that have been used. As they were designed for providing either an on or off result, they are unsuited for being switched on and off quickly. This means that there is a significant delay between the time they are enabled and the time that they begin to transmit a signal. For this reason, the enable lines in the program are not cleared once the echo has been received. For future applications, it is recommended that these sensors be replaced with others that are designed for a ranging application such as the Polaroid 3500 series of ultrasonic sensors.

CONCLUSION

The use of fuzzy logic in association with the ultrasonic sensors for intelligent obstacle avoidance has been implemented successfully and works with a good level of reliability. The Rug-Warrior Pro proved to be an excellent platform for the implementation of this project. With a few modifications, it was able to handle the requirements with ease, which also gives the potential for future modifications and development. The platform does, however, appear to possess a few issues with some of the expansion ports and also there is occasional unexplained freezing of the microcontroller. The power consumption of the entire package is one point of a possible problem; it will run effectively as it is but, with increased functionality, power drain may need to be considered. One solution may be to provide a long-life battery pack or perhaps a separate power supply for the ultrasonic sensors.

REFERENCES

1. I. Baturone *et al.*, *Microelectronic Design of Fuzzy Logic Based Systems*, CRC Press, Boca Raton (2000).
2. J. L. Jones, A. M. Flynn and B. A. Seiger, *Mobile Robots: Inspiration to Implementation* (2nd edition), A. K. Peters, Massachusetts (1999).
3. J. L. Jones, *Rug-Warrior Pro Assembly Guide*, A. K. Peters, Massachusetts.
4. A. Wright, R. Sargent and C. Witty, *Interactive C Users Guide* (0.9 ed.), Newton Research Labs (1997).
5. L. X. Wang, *A Course in Fuzzy Systems and Control*, Prentice-Hall International, New Jersey (1997).
6. W. L. Xu, S. K. Tso and Y. H. Fung, Fuzzy reactive control of a mobile robot incorporating a real/virtual target switching strategy, *Robotics and Automation Systems*, **23** (1998), pp. 171–186.
7. Velleman Components NV, *Velleman Parking Radar Manual* (http://www.velleman-kit.com/).
8. Motorola, *M68HC11E Family Data Sheet* (http://www.motorola.com/semiconductors/).

M. D. Hurley received a B.E. in Mechatronics with honours from Massey University, New Zealand, in May 2004. He now serves as an automation engineer in the New Zealand army.

W. L. Xu received a Ph.D. in Mechatronics and Robotics from Beijing University of Aeronautics and Astronautics, China, in 1988. He is a Senior Lecturer in Mechatronics at the Institute of Technology and Engineering, Massey University, Palmerston North, New Zealand. Prior to joining Massey in 1999, he worked at the City University of Hong Kong, the University of Stuttgart, Germany, and Southeast University, China. His current research interests include intelligent Mechatronics, advanced robotics and intelligent control. He is a senior member of IEEE and serves as Associate Editor for *IEEE Transactions on Industrial Electronics* and Regional Editor for the *International Journal of Intelligent Systems Technologies and Applications*.

Glen Bright graduated from the University of Natal, South Africa, with a Ph.D. in Mechatronics and Robotics in 1993. He currently leads the Mechatronics and Robotics Research Group (MR2G), which has active members in New Zealand, South Africa and North America. He is an active member of IEEE, ISPE, IASTED and a newly appointed CIRP member. His research interests include Internet Manufacturing, reconfigurable machines for Agile Manufacturing, Automated Guided Vehicle technologies and wireless communication for materials handling and automated machines.