FUZZY LOGIC CONTROL FOR ROBOT MAZE TRAVERSAL: AN UNDERGRADUATE CASE STUDY

James Wolfer[1], Chad A. George[2]

Abstract — As previously reported, Indiana University South Bend has deployed autonomous robots in their Computer Organization course to facilitate introducing computer science students to the basics of logic, embedded systems, and assembly language. The robots help to provide effective, real-time feedback on program operation and to make assembly language less abstract. As a part of their coursework students are required to program a sensor-based traversal of a maze. This paper details one solution to this problem employing a fuzzy logic controller to create linguistic rules.

Index Terms — Fuzzy logic, pedagogy, robots, student projects

.

INTRODUCTION

Assembly language programming in a computer science environment is often taught using abstract exercises to illustrate concepts and encourage student proficiency. To augment this approach we have elected to provide hands-on, real-world experience to our students by introducing robots into our assembly language class. Observing the physical action of robots can generate valuable feedback and have real-world consequences – robots hitting walls make students instantly aware of program errors, for example. It also provides insight into the realities of physical machines such as motor control, sensor calibration, and noise. To help provide a meaningful experience for our computer organization students, we reviewed the course with the following objectives in mind:

- Expand the experience of our students in a manner that enhances the student's insight, provides a hands-on, visual, environment for them to learn, and forms an integrated component for future classes.
- Remove some of the abstraction inherent in the assembly language class. Specifically, to help enhance the error detection environment.
- Provide a kinesthetic aspect to our pedagogy.
- Build student expertise early in their program that could lead to research projects and advanced classroom activities later in their program. Specifically, in this case, to build expertise to support later coursework in intelligent systems and robotics.

As one component in meeting these objectives we, in cooperation with the Computer Science department, the Intelligent Systems Laboratory, and the University Center for Excellence in Teaching, designed a robotics laboratory to support the assembly language portion of the computer organization class as described in [1].

The balance of this report describes one example project resulting from this environment. Specifically, we describe the results of a student project developing an assembly language fuzzy engine, membership function creation, fuzzy controller, and resulting robot behavior in a Linux-based environment. We also describe subsequent software devlopment in C# under Windows, including graphical membership tuning, real-time display of sensor activation, and fuzzy controller system response. Collectively these tools allow for robust controller development, assembly language support, and an environment suitable for effective classroom and public display.

BACKGROUND

Robots have long been recognized for their potential educational utility, with examples ranging from abstract, simulated, robots, such as Karel[2] and Turtle[3] for teaching programming and geometry respectively, to competitive events such as robotic soccer tournaments[4]. As the cost of robotics hardware has decreased their migration into the classroom has accelerated [5, 6]. Driven by the combined goals for this class and the future research objectives, as well as software availability, we chose to use off-the-shelf, Khepera II, robots from K-Team[7].

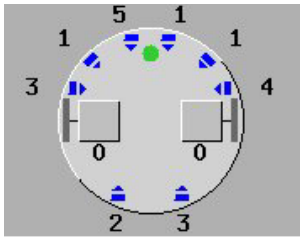

FIGURE 1

1 James Wolfer, Computer Science, Indiana University South Bend, South Bend, IN, USA,jwolfer@iusb.edu
2 Chad A. George, Computer Science, Indiana University South Bend, South Bend, IN, USA,cgeorge@cs.iusb.edu

SIMULATED ROBOT DIAGRAM

The K-Team Kephera II is a small, two-motor robot which uses differential wheel speed for steering. Figure 1 shows a functional diagram of the robot. In addition to the two motors it includes a series of eight infrared sensors, six along the "front" and two in the "back" of the robot. This robot also comes with an embedded system-call library, a variety of development tools, and the availability of several simulators. The embedded code in the Khepera robots includes a relatively simple, but adequate, command level interface which communicates with the host via a standard serial port. This allows students to write their programs using the host instruction set (Intel Pentium in this case), send commands, and receive responses such as sensor values, motor speed and relative wheel position.

We also chose to provide a Linux-based programming environment to our students by adapting and remastering the Knoppix Linux distribution [9]. Our custom distribution supplemented Knoppix with modified simulators for the Khepera, the interface library (including source code), manuals, and assembler documentation. Collectively, this provides a complete development platform.

The SIM Khepera simulator[8] includes source code in C, and provides a workable subset of the native robot command language. It also has the ability to redirect input and output to the physical robot from the graphics display. Figure 2 shows the simulated Khepera robot in a maze environment and Figure 3 shows an actual Khepera in a physical maze. To provide a seamless interface to the simulator and robots we modified the original simulator to more effectively communicate through a pair of Linux pipes, and we developed a small custom subroutine library callable from the student's assembly language programs.

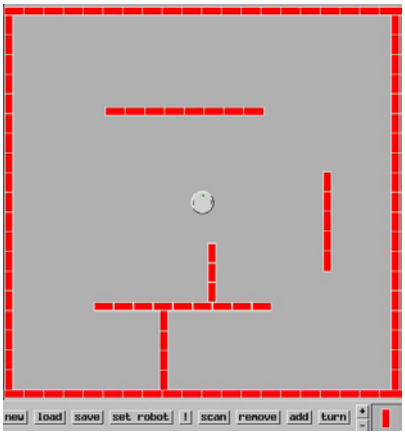

FIGURE 2
SIMULATED ROBOT IN MAZE

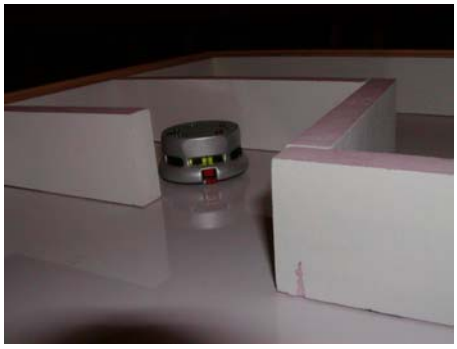

FIGURE 3
KHEPERA ROBOT IN MAZE

Assignments for the class range from initial C assignments to call the robot routines to assembly language assignments culminating in the robot traversing the maze.

FUZZY CONTROLLER

One approach to robot control, fuzzy logic, attempts to encapsulate important aspects of human decision making. By forming a representation tolerant of vague, imprecise, ambiguous, and perhaps missing information fuzzy logic enhances the ability to deal with real-world problems. Furthermore, by empirically modeling a system engineering experience and intuition can be incorporated into a final design.

Typical fuzzy controller design [10] consists of:

- Defining the control objectives and criteria
- Determining the input and output relationships
- Creating fuzzy membership functions, along with subsequent rules, to encapsulate a solution from intput to output.
- Apply necessary input/output conditioning
- Test, evaluate, and tune the resulting system.

Figure 4 illustrates the conversion from sensor input to a fuzzy-linguistic value. Given three fuzzy possibilities, 'too close', 'too far', and 'just right', along with a sensor reading we can ascertain the degree to which the sensor reading belongs to each of these fuzzy terms. Note that while Figure 4 illustrates a triangular membership set, trapezoids and other shapes are also common.

Once the inputs are mapped to their corresponding fuzzy sets the fuzzy attributes are used, expert system style, to trigger rules governing the consequent actions, in this case, of the robot. For example, a series of rules for a robot may include:

- If left-sensor is too close and right sensor is too far then turn right.
- If left sensor is just right and forward sensor is too far then drive straight.
- If left sensor is too far and forward sensor is too far then turn left.
- If forward sensor is close then turn right sharply.

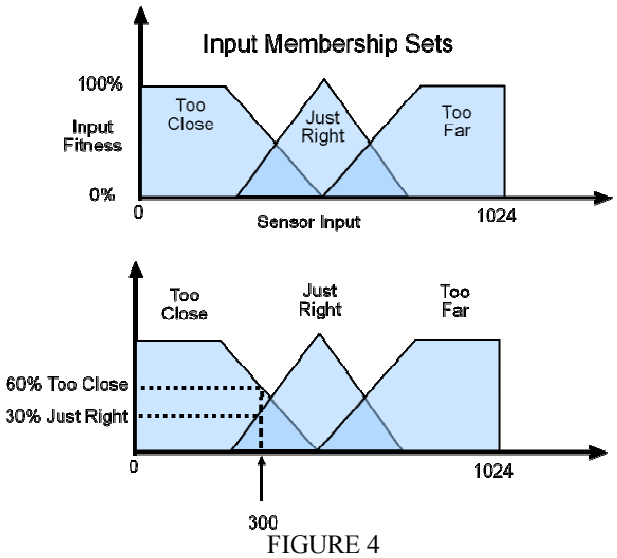

FIGURE 4
FUZZY INPUT MEMBERSHIP

The logical operators 'and', 'or', and 'not' are calculated as follows: 'and' represents set intersection and is calculated as the minimum value, 'or' is calculated as the maximum value or the union of the sets, and 'not' finds the inverse of the set, calculated as 1.0-fitness.

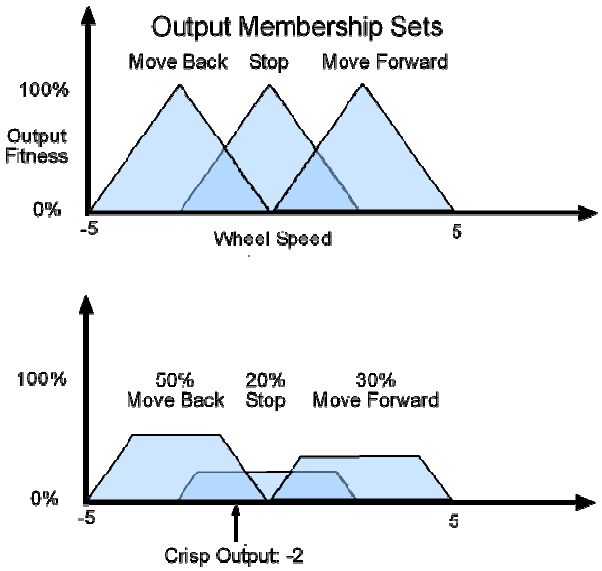

FIGURE 5
FUZZY OUTPUT AND DEFUZZIFICATION

Once inputs have been processed and rules applied, the resulting fuzzy actions must be mapped to real-world control outputs. Figure 5 illustrates this process. Here output is computed as the coordinate of the centroid of the aggregate area of the individual membership sets along the horizontal axis.

ASSEMBLY LANGUAGE IMPLEMENTATION

Two implementations of the fuzzy robot controller were produced. The first was written in assembly language for the Intel cpu architecture under the Linux operating system, the second in C# under Windows to provide a visually intuitive interface for membership set design and public demonstration.

Figure 6 shows an excerpt of pseudo-assembly language program. The actual program consists of approximately eight hundred lines of hand-coded assembly language. In the assembly language program subroutine calls are structured with parameters pushed onto the stack. Note that the code for pushing parameters has been edited from this example to conserve space and to illustrate the overall role of the controller. In this code-fragment the 'open_pipes' routine establishes contact with the simulator or robot. Once communication is established, a continous loop obtains sensor values, encodes them as fuzzy inputs, interprets them through the rule base to linguistic output members which are then converted to control outputs which are sent to the robot. The bulk of the remaining code implements the fuzzy engine itself.

```
_start:
        call open_pipes

_main_loop:

        # Read the sensor values from the robot
        call _read_sensors

        # Fuzzify the crisp inputs
        call _fuzzify_input

        # Reset the output fitness
        call _reset_output

        # Map input fitness values to output fitness values
        call _logic_engine

# Convert output fitness values to crisp motor values
        call _defuzzify_output

# Send motor commands to robot
                call _move_robot

        jmp _main_loop
```

FIGURE 6

FUZZY CONTROLLER MAIN LOOP

Membership sets were manually defined to allow the robot to detect and track walls, avoid barriers, and negotiate void spaces in it field of operation. Using this controller, both the simulated robot and the actual Khepera successfully traversed a variety of maze configurations.

ASSEMBLY LANGUAGE OBSERVATIONS

While implementing the input fuzzification and output defuzzification in assembly language was tedious compared with the same task in a high level language, the logic engine proved to be well suited to description in assembly language.

The logic rules were defined in a type of psuedo-code using 'and', 'or', 'not' as operators and using the fuzzy input and output membership sets as parameters. With the addition of input, output and flow control operators, the assembly language logic engine simply had to evaluate these psuedo-code expressions in order to map fuzzy inputs memberships to fuzzy output memberships.

Other than storing the current membership fitness values from the input fuzzyfication, the only data structure needed for the logic engine is a stack to hold intermediate calculations. This is convenient under assembly language since the CPUs stack is immediately available as well as the nescesary stack operators.

There were seven commands implemented by the logic rule interpreter: IN, OUT, AND, OR, NOT, DONE, and EXIT.

- IN – reads the current fitness from an input membership set and places the value on the stack.
- OUT – assigns the value on the top of the stack as the fitness value of an output membership set if it is greater than the existing fitness value for that set.
- AND – performs the intersection operation by replacing the top two elements on the stack with the minimum element.
- OR – performs the union operation by replace the top two elements on the stack with their maximum.
- NOT – replaces the top value on the stack with its compliment.
- DONE – pops the top value off the stack to prepare for the next rule
- EXIT – signals the end of the logic rule definition and exits the interpreter.

As an example the logic rule "*If left-sensor is too close and right sensor is too far then turn right*", might be defined by the following fuzzy logic psuedo-code:

```
IN, left_sensor[ TOO_CLOSE ]
IN, right_sensor[ TOO_FAR ]
AND
OUT, left_wheel[ FWD ]
OUT, right_wheel[ STOP ]
DONE
EXIT
```

By utilizing the existing CPU stack and implementing the logic engine as an psuedo-code interpreter, the assembly language version is capable of handling arbitrarily complicated fuzzy rules composed of the simple logical operators provided.

C# IMPLEMENTATION

While the assembly language programming was the original focus of the project, ultimately we felt that a more polished user interface was desirable for membership set design, fuzzy rule definition, and controller response monitoring. To provide these facilities the fuzzy controller was reimplemented in C# under Windows.

Figures 7 through 10 illustrate the capabilities of the resulting software. Specifically, Figure 7 illustrates user interface for membership defination, in this case 'near'. Figure 8 illustrates the interface for defining the actual fuzzy rules. Figure 9 profiles the output response with respect to a series of simulated inputs. Finally, real-time monitoring of the system is also implemented as illustrated in 10 which shows the robot sensor input values.

Since the Khepera simulator was operating system specific, the C# program controls the robot directly. Again, the robot was successful at navigating the maze using a controller specified with this interface.

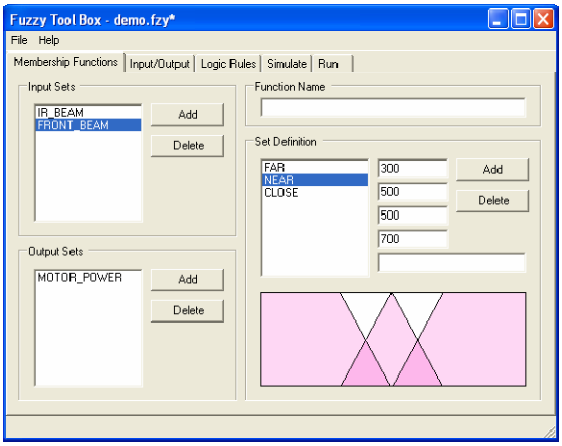

FIGURE 7
KHEPERA SENSOR RESPONSE

SUMMARY

To summarize, we have developed a student-centric development environment for teaching assembly language programming. As one illustration of its potential we profiled a project implementing a fuzzy-logic engine and controller, along with a subsequent implementation in the C# programming language. Together these projects help to illustrate the viability of a robot-enhanced environment for assembly language programming.

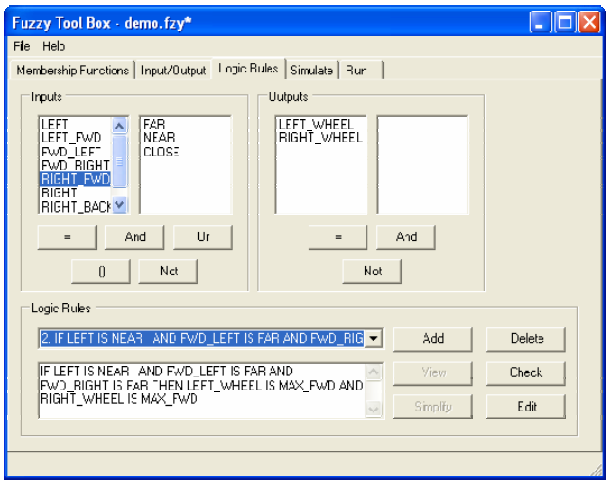

FIGURE 8
FUZZY RULE DEFINITION

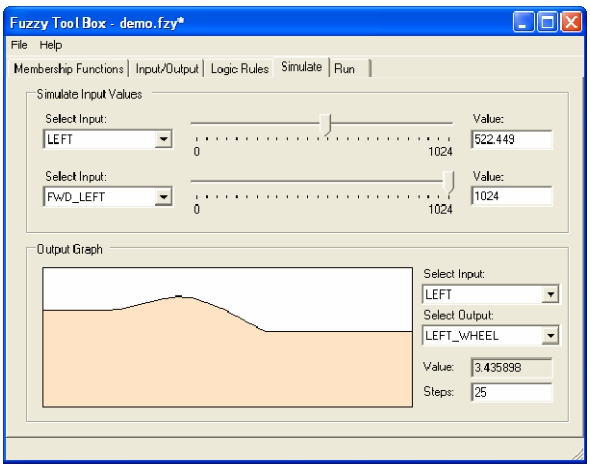

FIGURE 9
LEFT WHEEL OUTPUT PROFILE

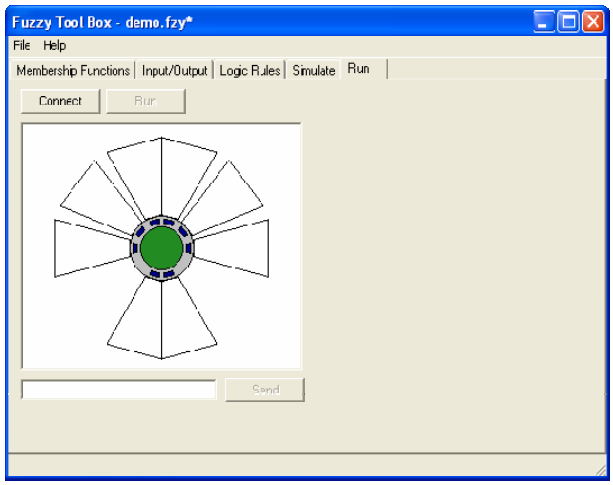

FIGURE 10
KHEPERA SENSOR RESPONSE

REFERENCES

[1] Wolfer, J & Rababaah, H. R. A., "Creating a Hands-On Robot Environment for Teaching Assembly Language Programming", *Global Conference on Engineering and Technology Education*, 2005

[2] Pattic R.E., Karel the Robot: a gentle introduction to the art of programming, 2nd edition. *Wiley*, 1994

[3] Abelson H. and diSessa A., Turtle geometry: the computer as a medium for exploring mathematics. *MIT Press*, 1996

[4] Amirijoo M., Tesanovic A., and Nadjm-Tehrani S., "Raising motivation in real-time laboratories: the soccer scenario" in *SIGCSE Technical Symposium on Computer Sciences Education*, pp. 265-269, 2004.

[5] Epp E.C., "Robot control and embedded systems on inexpensive linux platforms workshop," in *SIGCSE Technical Symposium on Computer Science Education*, p. 505, 2004

[6] Fagin B. and Merkle L., "Measuring the effectiveness of robots in teaching computer science," in *SIGCSE Technical Symposium on Computer Science Education*, PP. 307-311, 2003.

[7] K-Team Khepera Robots, http://www.k-team.com, accessed 09/06/05.

[8] Michel O., "Khepera Simulator package version 2.0: Freeware mobile robot simulator written at the university of nice Sophia-Antipolis by Olivier Michel. Downloadable from the world wide web. http://diwww.epfl.ch/lami/team/michel/khep-sim, accessed 09/06/05.

[9] Knoppix Official Site, http://www.knoppix.net, accessed 09/06/05.

[10] Earl Cox., *The Fuzzy Systems Handbook*, Academic Press, New York, 1999.