

# Neural Network Control for A Fire-Fighting Robot

Yan Zhou, Dawn Wilkins, Robert P. Cook

University of Mississippi, Dept. of Computer and Information Science, Weir 302, University, MS 38677  
e-mail: zy, dwilkins, bobcook@olemiss.edu

**Abstract.** The paper discusses the development of an associative, neural network as an on-line algorithm to train and control a fire-fighting robot. Learning is externally supervised with encoded target actions. The robot acquires basic navigation skills as well as the ability to detect a fire and to extinguish it.

---

**Keywords:** robotics, neural network, control algorithm, fire fighting.

---

## 1. Introduction

The goal of this paper is to present the design of a fire-fighting robot (Daisy) that was developed to fulfill the requirements for a Robotics course at the University of Mississippi. The robot had to move forward from a starting position, navigate around obstacles, detect any fires (birthday candles in this experiment) in its path, and then extinguish them.

We review the attributes of the controller board, and the robot's mechanical and sensor system. Next, the neural network control algorithm is discussed, together with the training procedure. Finally, we summarize the results of several experiments that measured the robot's fire-fighting ability.

## 2. Background

The robot is constructed from several Lego Technic™ kits that included beams, connectors, motors, gears, axles and the other parts necessary for the mechanical design. The controller is Martin's [1] Handy Board system.

The Handy Board features a Motorola 6811 microprocessor, 32KB of battery-backed CMOS static RAM, motor driver chips, a two-line, 16 character LCD display, internal 9.6v DC NICAD battery with a built-in recharging circuit, serial PC interface, 7 analog inputs, 9 digital inputs, a beeper and several buttons. The processor is programmed using Interactive C[2], which is a

subset of C that supports an interactive PC interface to a Handy Board that is connected with a serial cable.

The compiler generates a p-code variant for a stack-machine architecture. The run-time library is a subset of the C library, but also includes floating-point emulation, device drivers, and a concurrent programming kernel. Programs are compiled on a host PC and then downloaded to the board for execution and debugging.

Once a program is downloaded, the board can be turned off and attached to a robot chassis for autonomous operation. The integrated battery pack supplies power to maintain the content of the static RAM and to drive the motors and sensors when the robot is operational. As long as the battery power remains sufficient, the Handy Board can be turned off and on any number of times to repeat a program.

During an operational test, there are several debugging strategies to compensate for the absence of a PC control console. First, assert statements can be used liberally to document program assumptions. Any violation at runtime will be displayed on the LCD display. Second, Interactive C implements **persistent** variables that retain their values on power off. They are valuable when used to implement a small circular buffer to record inputs and state transitions. After a failure, the Handy Board's buffer can be up-loaded to a PC for examination. Finally, the beeper and the LCD can be used to signal state transitions to the operator. We found it useful to write a simple routine that allows each of the 32 character positions on the LCD to be treated as a separate output stream. Another option uses a separate process to time multiplex the LCD among N message streams.

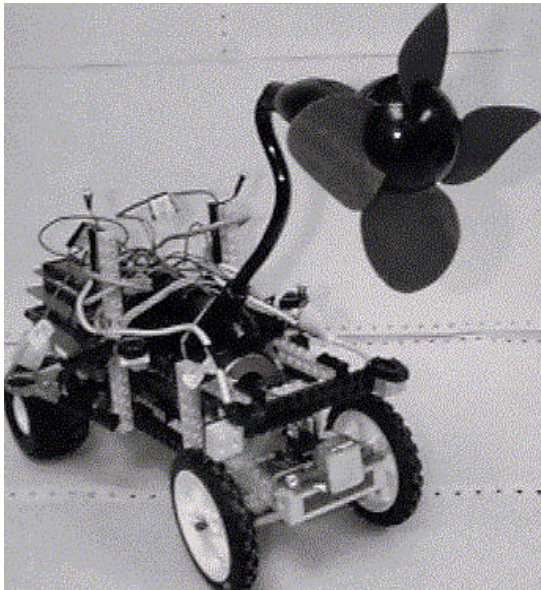
## 3. Mechanical/Sensor Design

Daisy is a four-wheel vehicle with a rear-wheel, bi-directional drive and front-wheel steering. Figure 1

illustrates the final design. The construction was facilitated by Martin's paper [3] on Lego™ design techniques.

The frame of the robot consists of the minimum number of beams necessary to support the weight of the Handy Board and the motors. Any extra weight would result in an added drain on the battery; thus, reducing a mission's lifetime. Overall, the robot is thirty centimeters long, eighteen wide, and seventeen centimeters tall, not including the fan. The fan is attached by a flexible arm, and is positioned twenty-two centimeters from the floor.

A single DC motor is used to drive the rear wheels. Gear reduction produces the torque needed to propel Daisy at a steady, but slow, speed. The number of gears was minimized in order to reduce frictional losses. The front-wheel steering mechanism also uses gear reduction, but for a stepper motor, to control Daisy's direction. A stepper motor can regulate the shaft's rotational position based on control signals.

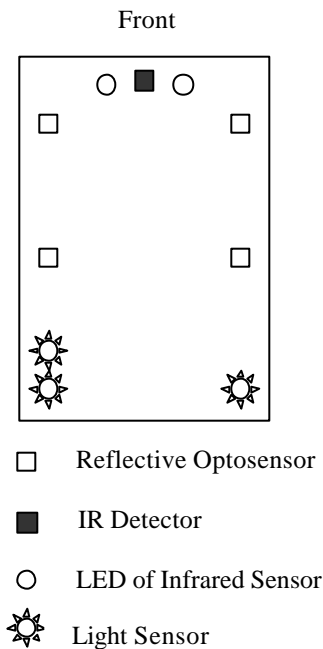


**Figure 1. Daisy, the Robot**

Initially, it was decided that the most difficult physical problem would be extinguishing a fire. The proposed solution was to use a squirt bottle. The handle would be controlled using Flexinol™ muscle wire. By applying current, the wire could be caused to contract; on removing the current, it would expand. Multiple wires were to be used in

parallel to produce the necessary force. Unfortunately, the wire was very fine (50µm), which presented anchoring and handling problems. Higher diameter wires would have drawn too much current to be feasible.

The second solution was to use an electric fan that was to be turned on in ten-second bursts. The burst behavior was necessary to give a fire an opportunity to rekindle if it was not totally extinguished. The fan worked satisfactorily. (One advantage of using a fan was that blowing air on bystanders during the robot's training was more socially acceptable than squirting them.)



**Figure 2. Sensor Placement**

Figure 2 illustrates the sensor placement on Daisy. The front of the vehicle contains an infrared proximity detector. It has two emitters (LEDs) and one detector. It uses modulated IR to detect obstacles at distances from 10 to 30 centimeters. The two emitters are used alternatively, and therefore can share a single detector. We covered the detector with layers of film to reduce its sensitivity to twelve centimeters; thus, Daisy could be programmed to drive closer to an obstacle before turning.

The four reflective optosensors combine an IR emitter and detector in a single device. They were used to detect fire sources, which were birthday candles in our experiments. The optosensors

seemed to have significant variance in their sensitivity, but detected fire, on average, from a distance of approximately one meter. The peripheral range of the optosensors was approximately  $40^\circ$  centered about straight ahead. The best height for detection of a flame was seven centimeters above the floor.

The three light sensors (Figure 2) at the top of Daisy comprise the "brain". One of the unique aspects of our project is that Daisy is trained "in situ". Thus, the first sensor is used to differentiate between training mode and operation. The size of the light sensors made it feasible to cover them with ballpoint pen caps to signal a zero or a one.

For simplicity, Daisy's training actions were unary; that is, either forward, or left, or right, or blow; but no two simultaneously. As a result, two inputs were sufficient to indicate to Daisy the action that was desired in a particular state.

Several problems were initially encountered if Daisy moved too quickly. First, the human controller had to chase the robot and secondly, while doing so, it was quite difficult to get a cap on the correct photodiode. The solution was to gear down the speed. This had the further benefit of increasing the precision of the training since new inputs were occurring at a longer interval than the human reaction time.

In the following sections, we introduce the concept of neural net control and we explain how the sensor inputs are mapped in the control algorithm. Also, the training methodology and results are reviewed. Finally, the results of Daisy's fire-control experiments are presented and we conclude with a discussion of the results.

## 4. Neural Network Controller

A **neural network** is a set of nodes, called **units**, which are connected by **links**. A numeric value, called a **weight**, is associated with each link. All weights in a network are generally initialized to a random value between zero and one. Learning in a neural network is typically performed by adjusting the weights. The weights also serve as the memory of the network. Some of the units in the network interface with the environment, and therefore serve as input or output units. In addition, many neural networks have one or more layers of hidden units that do not have direct

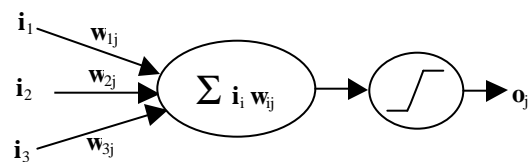
contact with the environment. Such networks are sometimes called multi-layer networks.

For the experiments, we decided to use a perceptron network, which is a very simple neural network that has only input and output units. The values of the input units of a perceptron network are directly mapped to the output values [4]. Learning with a perceptron network is simpler and much faster than with a multi-layer network, but perceptron networks are limited in what they can represent.

There are two phases associated with using a neural network for robotics [5]. First, the network must be trained, then it can be tested or used. Training a network is accomplished by adjusting the weights associated with the links in order to map each input pattern to the desired output pattern for that input. That is, a function must be learned that performs this mapping. The function is represented by the weights associated with the links. Once the network has been trained, the function is evaluated by processing the inputs (without updating the weights), and taking the action corresponding to the output unit with the largest value.

A single perceptron unit **j** is shown in Figure 3. The unit computes a weighted sum of its inputs, and then transforms the weighted sum into the output value for the unit,  $o_j$ , by passing it through an activation function. Many types of activation functions can be used.

The most common activation functions are the step function, sign function, and Sigmoid function. The step function outputs a one when the weighted sum exceeds some predefined threshold and a zero otherwise. The sign function is a step function with a threshold equal to zero. The Sigmoid function takes values from a possibly large input domain and maps them to values between zero and one. For this reason, the Sigmoid function is often called a squashing function.



**Figure 3. A Unit in a Perceptron Network**

In order to update the weights, a form of back-propagation is used. Back-propagation essentially performs a gradient-descent search to find the set of weights that best map the inputs to the correct, or target, outputs. The best mapping is the one with the least total error.

If the output from unit  $j$  is  $o_j$ , and the correct output for the unit is  $t_j$ , then the error is  $t_j - o_j$ . If the error is positive, then the output from unit  $j$  needs to be greater. If the error is negative, then the output from unit  $j$  needs to be decreased. This result can be achieved by increasing or decreasing the weights of the links coming into unit  $j$  (depending on whether the current weight of a link is positive or negative).

Normally a constant  $h$ , the learning rate, is defined to determine how quickly learning should occur (that is, how large the steps should be in the search for the best set of weights). If  $h$  is too small, the training could take a long time. If it is too large, there is a risk of overstepping the best set of weights. In addition to a learning rate term, it is often useful to include a momentum term,  $a$ , that gives extra weight to continue the search in the same direction. This term helps avoid useless oscillation during the search. Training is discussed in more detail in Section 4.2.

### 4.1 Daisy's Network

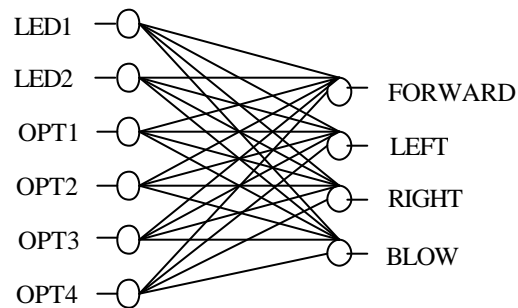
Daisy, our fire-extinguishing robot, has a limited amount of memory and must operate in a real-time environment. Thus, it was decided to equip her with a perceptron network. This type of network is known to have limited representation power (e.g., non-linearly separable functions such as exclusive-or cannot be represented exactly), but this limitation has not been a problem in our experiments. Nehmzow and McGonigle [6] have also obtained very good results using this type of network.

Figure 4 shows the architecture of the network. There are six input units—four units correspond to the reflective optosensors (OPT1-OPT4) which are used for detecting fire, and two units (LED1

and LED2) are associated with the proximity detection sensor, which is used for navigation. There are four output units that control the four possible actions Daisy may take: **FORWARD**, **TURN\_LEFT**, **TURN\_RIGHT** and **BLOW**.

Since the input values are read from different types of environmental stimuli, they are normalized to a real number between zero and one so that an individual sensor does not dominate the others. This is called sensor fusion.

The activation unit used in Daisy is a Sigmoid function. Thus the result from each output unit is a continuous value (rather than a binary zero or one). Daisy is programmed to take the action that corresponds to the output unit having the highest value.



**Figure 4. Daisy's Neural Network**

Training Daisy is accomplished by providing external feedback, similar to “shaping” as described by Nehmzow and McGonigle [6]. Feedback is given to Daisy using the three light sensors (see Figures 1, 2). One of the light sensors controls whether or not the robot is in training mode. If already trained, each input produces an output (action), but weights are not updated. When training, the weights are updated when the input is processed. The remaining two light sensors are used to encode the target action. Table 1 shows the details of how the desired actions are communicated to Daisy.

Desired Action	Light Sensor Encoding	Target vector $t$
----------------	-----------------------	-------------------

<b>FORWARD</b>	Both sensors Covered	{1, 0, 0, 0}
<b>TURN_LEFT</b>	Left uncovered Right covered	{0, 1, 0, 0}
<b>TURN_RIGHT</b>	Left covered Right uncovered	{0, 0, 1, 0}
<b>BLOW</b>	Both sensors Uncovered	{0, 0, 0, 1}

**Table 1. Encoding of Target Actions**

Daisy has been programmed to simply move forward on a **FORWARD** action. **TURN\_LEFT** consists of turning the front wheels to the right, reversing for 1 second, stopping, turning the front wheels to the left, moving forward for one-half second, stopping, straightening the front wheels, and then proceeding forward. The **TURN\_RIGHT** action is programmed similarly. The timing for the turn sequences was determined by trial-and-error.

For the **BLOW** action, Daisy first positions herself to face the fire head-on (that is, to ensure the fire is detected by one of the front two optosensors), and then the fan is engaged for ten seconds. If the fire has not been extinguished, the robot will move forward for 100 milliseconds and initiate the fan again. This procedure is repeated until the fire is no longer detected. At that point, the robot will resume normal navigation.

## 4.2 Training

The goal of training the network is to adjust the weights so that the input patterns (sensor inputs) are mapped to an output pattern that corresponds to the “correct” action Daisy should take. Daisy is trained by simulating the situations that she can expect to encounter. The trainer places her hand in front of an infrared sensor, say on the left side, and then gives the target action **TURN\_RIGHT**, and vice versa. When none of the sensors detects an obstacle, the target action **FORWARD** is given. When a candle is placed within sensor range, Daisy is given feedback to **BLOW**.

Initially, all weights in the network are assigned random values (between zero and one). During training, when an incorrect action is taken by the robot, the correct (target) action is indicated by covering, or uncovering, the light sensors as described in Table 1. Weights are updated during training using simple back-propagation of errors.

The formula for updating  $w_{ij}$ , the weight of the link between input unit  $i$  and output unit  $j$ , at time  $t+1$  is:

$$w_{ij}(t+1) = w_{ij}(t) + h \cdot (t_j(t) - o_j(t)) \cdot i_i(t) + a \cdot \Delta w_{ij}(t-1),$$

where  $h$  is the learning rate (defined as 0.3),  $t_j(t)$  and  $o_j(t)$  are the target output and actual output from unit  $j$ , respectively at time  $t$ ,  $i_i(t)$  is the input at unit  $i$  at time  $t$ ,  $a$  is the learning momentum (also defined as 0.3), and  $\Delta w_{ij}(t-1)$  is the weight update increment on the link from unit  $i$  to unit  $j$  in the previous iteration.

One difficulty faced was to train the robot to distinguish between a lit candle and an obstacle in the environment. This was solved by disabling the infrared sensor when the optosensors detected a fire ahead. Once a fire was no longer detected, the infrared sensor was enabled for normal navigation. Another problem encountered was physically reacting quickly enough to communicate the training signal. The light sensors are small and managing the ballpoint pen caps was challenging. The slight delay caused by the process did not seem to significantly affect the training outcome.

The initial attempt at training proceeded in a sequential fashion. First, training for navigation (obstacle avoidance) was performed, followed by training for fire detecting and extinguishing. Unfortunately, this method was not very productive. Training on the second task was difficult, and when reasonably successful, the first task seemed to be “unlearned”. This problem was probably due to overfitting the weights of the network to the most recent task.

A second attempt at training Daisy produced much better results. This time the robot was trained on navigation and fire-fighting at the same time (alternating between simulating the various tasks).

At this point, the training of the network occurred very quickly. Only a few examples of feedback for each operation were required. The process required just a few minutes of real time to complete.

Figure 5 shows a sample progression of the changes in the 24 link weights during one training

session. The first four weights correspond to the links from input unit one to output units one through four, respectively. The next four weights correspond to the links from input unit two to the four output units, and so on. The initial weights, indicated by the circles are random values between zero and one.

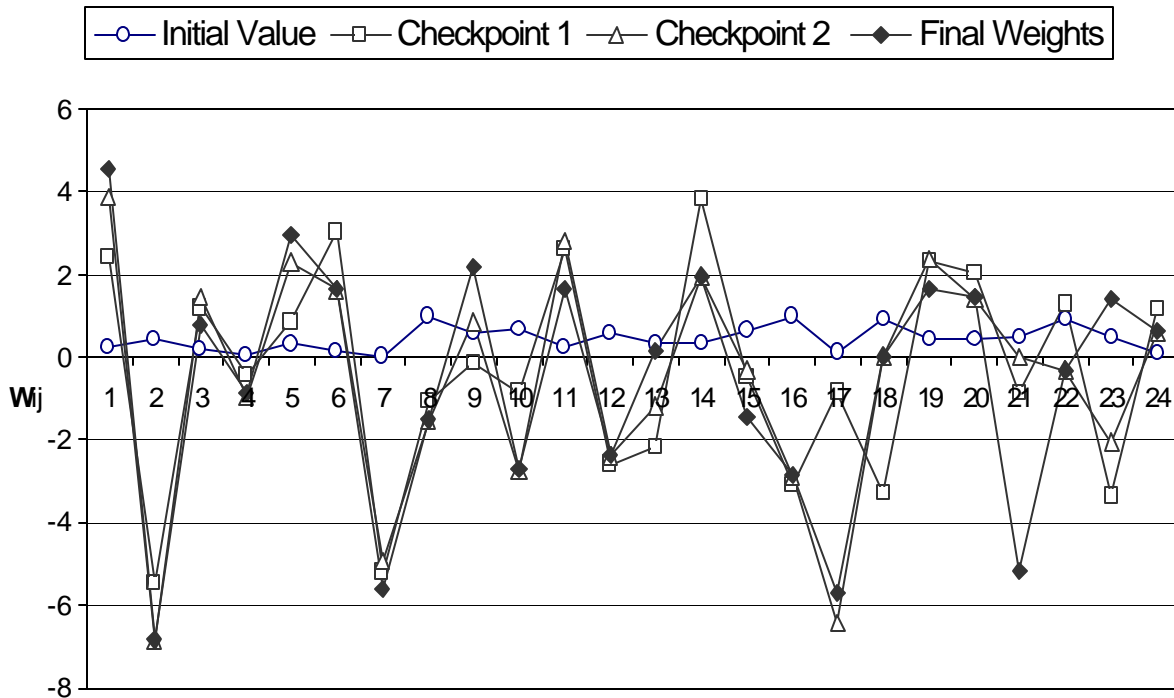
The weights in the network are shown at two intermediate checkpoints and at their final values. At Checkpoint 1, Daisy had learned to navigate and blow, but not to detect fire. By Checkpoint 2, Daisy had also learned to detect a fire on the left-hand side.

Although the weights had no physical meaning, it was clear that, in this case, the weights on the links associated with the first two sensors (LED1 and LED2) converged quickly. This seemed to indicate that the first tasks actually learned by the robot related to navigation. The next weights to converge corresponded to OPT1 and OPT2, the two optosensors nearest the front of the robot. Finally, the weights of the links associated with sensors OPT3 and OPT4 converged. Thus, in this

training session, fire detection in the periphery was the final task learned.

Once the neural net had been trained, no additional training was necessary unless new tasks were to be learned. If there were additional tasks the robot needed to accomplish, more training time would be required. Note that Daisy was not trained to a particular environment, but rather to accomplish tasks.

With an integrated training strategy, the robot successfully learned both the obstacle navigation and fire detecting and extinguishing tasks. Because of the poor results in the first attempt at training, it seemed that the performance of the real-time neural network was very sensitive to the input sequence of the training examples. Results from testing the robot are given in the next section. The goal is to have Daisy achieve her objectives, even when confronted with different environments.



$i = 1 \text{ to } 6; j = 1 \text{ to } 4$

**Figure 5. Convergence of 24 Weights**

## 5. Experiments

Two sets of experiments were run using the final weights shown in Figure 5. The first experiment consisted of five tests (A, B, C, D, and E) in an enclosed 1.5 x 2 meter, obstacle-free area. A single candle was placed in one corner of the area. In each run, the robot was placed in a different location, facing away from the wall. The floor plan and starting locations of the robot are shown in Figure 6.

The starting locations of the robot for the five experiments are indicated by A, B, C, D, and E in the figure. The candle is indicated by the ☼ symbol. Daisy was able to locate and extinguish the fire in each test. Table 2 shows the time required to find and extinguish the fire in each case.

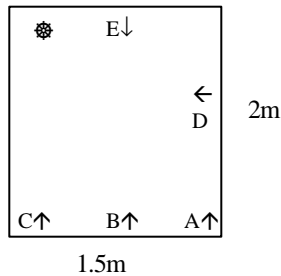


Figure 6. Experiment, No Obstacles

Experiment	Time required
A	1 minute
B	1 minute
C	2 minutes
D	5 minutes
E	4 minutes

Table 2. Timings for First Experiment

The second set of experiments consisted of five runs, similar to those of the first experiment, except in this case, the layout included two candles and three obstacles. The floor plan is shown in Figure 7, and the results are presented in Table 3. The column labeled “Candle 1” in Table 3 lists the amount of time from the beginning of the experiment until the first of the candles is extinguished. The column labeled “Candle 2” is

the amount of time (starting at zero) from the time the first candle is extinguished until the second one is extinguished. The robot was not repositioned after extinguishing the first candle.

During the second (B) run, Daisy found and extinguished one candle, then located the second candle. While repositioning for the **BLOW** action, the robot became oriented so that the candle was

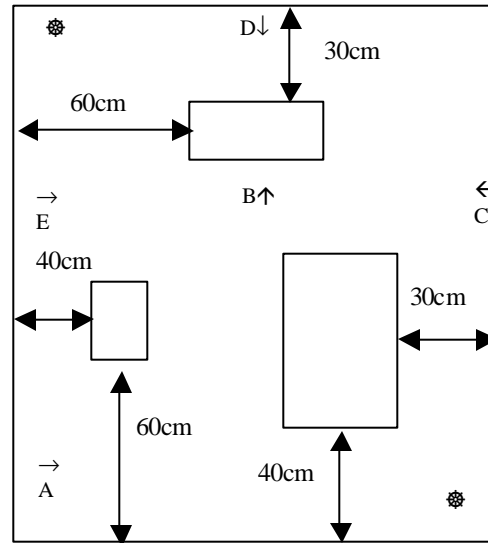


Figure 7. Experiment, With Obstacles

no longer detectable by the sensors, and Daisy resumed navigation mode. Once back in navigation mode, the robot drove into a different corner and became stuck. She continued to maneuver, but was oscillating between detecting a left and right obstacle (the corner) and was making extremely slow progress. After 10 minutes, the run was abandoned.

Experiment	Candle 1	Candle 2
A	2 minutes	4 minutes
B	11 minutes	(abandoned)
C	2 minutes	7 minutes
D	2 minutes	7 minutes
E	2 minutes	5 minutes

Table 3. Timings for Obstacle Experiment

The oscillation problem was later corrected by increasing the distance Daisy reversed in proportion to the number of left turns that were made in the last ten seconds. This approach

enabled the robot to turn more degrees in one direction than in the other when it was stuck in a corner.

## 6. Discussion

In general, Daisy has no problem in detecting and extinguishing candles. The robot does not move rapidly, which is reflected somewhat in the timings. Once a flame is within range of the detectors, the robot is quite efficient. Otherwise, it tends to wander in an aimless fashion. Sensors that work at a greater range would help alleviate this problem. In addition, the infrared sensor is much more sensitive to white objects than to dark-colored objects. Having all light-colored obstacles yields much more consistent performance.

Overall, the results were very good. Training was fast and the neural network performed well. The robot did not perform spurious **BLOW** actions, and maneuvered fairly easily through any layout of obstacles without additional training.

There are a number of interesting extensions to this work. Adding more sensors, especially to the rear of the robot, would be of great help when reversing direction. In our experiments, we never had a situation where the robot backed into a candle, but it is not impossible.

Extra sensors would also be useful in detecting flames at various heights. Our sensors required a fairly small range in flame height to be effective. Adding a more powerful fan would speed the process of extinguishing the flame (it often took up to 30 seconds). The problem is that a more powerful fan would also require extra power that is not readily available.

Other paths to pursue are to track and extinguish moving flames, and to add a “virtual nose”, that is, a sensor to detect the odor of the fire (for example, see Keller [7]). This might reduce the time necessary to locate the fire, making extinguishing it easier.

*Acknowledgements:* The authors wish to thank Tim Bryant and Chris Riley for designing the Lego prototype of the robot and for implementing the circuit to drive the stepper motor.

## References

1. Martin, F., <http://lcs.www.media.mit.edu/groups/el/projects/handy-board/>.
2. Martin, F., Interactive C Manual for the Handy Board, <http://lcs.www.media.mit.edu/groups/el/projects/handy-board/software/icmanual/icmain.html>.
3. Martin, F., The Art of LEGO Design, **The Robotics Practitioner** 1, 2(1995) 1-18.
4. Rumelhart, D.E., G.E. Hinton and R.J. Williams, Learning Internal Representations by Error Propagation, **Parallel Distributed Processing** (Volume 1), D.E. Rumelhart and J.L. McClelland (Eds), Cambridge, MA, MIT Press (1986).
5. Zalzal A.M.S. and A.S. Morris, **Neural Networks for Robotic Control: Theory and Applications**, Ellis Horwood, New York (1996).
6. Nehmzow, U. and B. McGonigle, Achieving Rapid Adaptations in Robots by Means of External Tuition, **From Animals to Animats: Proceedings of the Third International Conference on Simulation of Adaptive Behavior**, D. Cliff, P. Husbands, J. Meyer and S. W. Wilson, (1994) 301-308.
7. Keller, Patrick, Institut Biomedizinische Technik, [http://www.ibmt.fhg.de/public\\_e/Produktblaetter/pages/infonose.htm](http://www.ibmt.fhg.de/public_e/Produktblaetter/pages/infonose.htm), (May 19, 1998)