Machine Learning, 31, 201–222 (1998)

Training a Vision Guided Mobile Robot

GORDON WYETH wyeth@csee.uq.edu.au
Department of Computer Science and Electrical Engineering, The University of Queensland, Brisbane, QLD 4072, Australia

Editors: Henry Hexmoor and Maja Matarić

Abstract. This paper presents the design, implementation and evaluation of a trainable vision guided mobile robot. The robot, CORGI, has a CCD camera as its only sensor which it is trained to use for a variety of tasks. The techniques used for training and the choice of natural light vision as the primary sensor makes the methodology immediately applicable to tasks such as trash collection or fruit picking. For example, the robot is readily trained to perform a ball finding task which involves avoiding obstacles and aligning with tennis balls. The robot is able to move at speeds up to 0.8 ms^{-1} while performing this task, and has never had a collision in the trained environment. It can process video and update the actuators at 11 Hz using a single $20 microprocessor to perform all computation. Further results are shown to evaluate the system for generalization across unseen domains, fault tolerance and dynamic environments.

Keywords: mobile robots, neural networks, machine vision, robot learning

1. Introduction

CORGI is a visually guided robot dog that has been trained to perceive and act in an office environment. The techniques used for developing CORGI's perception and action are designed to be applicable to a wide variety of tasks: cleaning the house, collecting trash from our parks and waterways, picking fruit from an orchard or collecting rocks on interplanetary exploration missions. The generic nature of the techniques comes from several factors.

- The robot is trained to perform perception and action, rather than programmed;
- the robot uses natural light vision which can be applied to many tasks in our vision oriented world;
- the model used for control has been kept strictly minimalist, based on the belief that simplicity should lead to a more generally applicable solution.

CORGI, the test bed for these techniques, is a completely autonomous robot that performs the learned tasks in real time (meaning human-like reaction speed). All computation, including vision, is carried out on a single, commonly available $20 microprocessor. The robot is about 30 cm long and stands 15 cm tall, with its CCD camera mounted on a manipulator at the front of the robot (Figure 1). These physical attributes give the robot the appearance of a small dog - hence the name, CORGI.

Robot training implies that the robot uses supervized learning to gain competence in its task. That is, a human operator controls the robot for a period of time to demonstrate the desired behavior of the robot in typical situations. The robot's learning task is not to duplicate the human controller's action, but to find the salient parts of the sensor to actuator

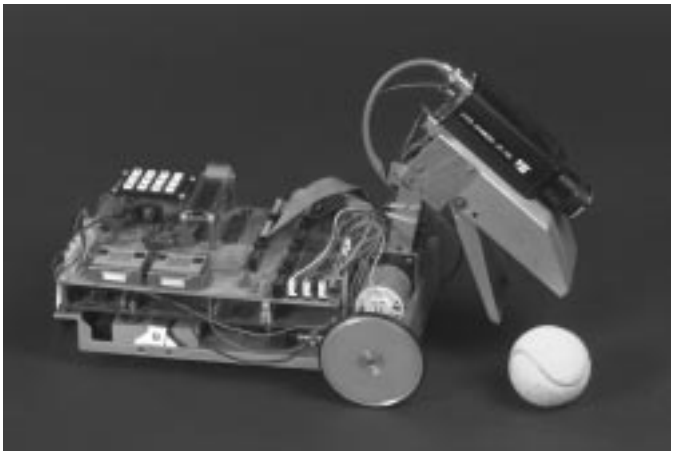

Figure 1. CORGI is a demonstration of the methods for robot perception and control described in this paper. The robot measures about 450 mm from nose to tail, and about 200 mm across the drive wheels. Its resemblance to a small dog led to its name.

relationship in order to produce reasonable behavior in both learned and novel situations. An operator can demonstrate only a very small proportion of the possible conditions that the robot will encounter; the possible positions and orientations of the robot in the environment are in no way discretized, and noisy sensor data implies that sensor readings will not be the same at a given position and orientation anyway. Add to this problem that real environments are dynamic and it becomes clear that the number of novel situations grossly exceeds the number of practicable training examples.

This argument precludes the approach used to train the ALVINN system (Pomerleau, 1993) to steer along a variety of road surfaces by a human operator. ALVINN used raw video as input and generated steering angles as output. In between was a single multilayer neural network which was trained using the well known backpropagation algorithm (Hertz et. al., 1991). Roads provide a much less diverse input than the environments which can be envisaged for robot operation. Also, steering is a relatively simple behavior compared to the complex behavior required of the robots described above.

The approach used in this paper is to split perception and action into distinct modules, with careful attention to the interface in between. The difficult perception training task can be performed off board the robot, reducing the processing power required for the robot. Then, with the trained perception system running on board, the robot behavior can be trained in real time, ensuring that the complex coordination of perception and action is achieved as well as possible.

The key to making this system function successfully is choosing the interface. The interface chosen is inspired by the vehicles described by (Braitenberg, 1984). Braitenberg describes a series of vehicles that demonstrate how simple structures of nerve-like components attached to sensors and actuators can create behavior that appears intelligent. The first few of these vehicles are the most popularized, featuring in a popular science journal

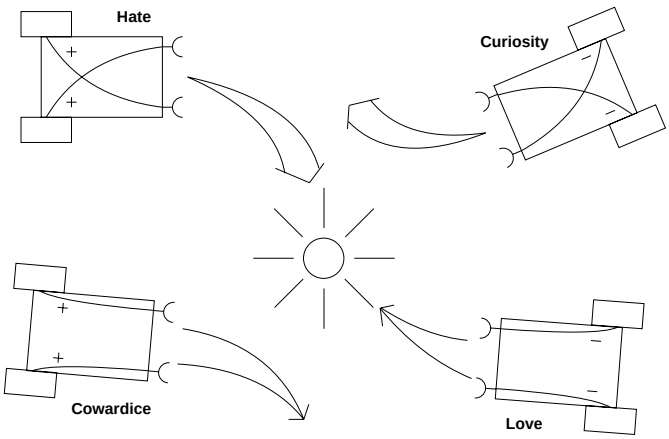

Figure 2. The four basic Braitenberg vehicles can demonstrate useful behavior towards a light source from very simple connection schemes. The arcs at the front of the vehicles are sensors. The sensors connect to the drive units through connections that may be crossed or uncrossed, excitory (+) or inhibitory (-). The labeled emotions are terms Braitenberg uses to describe the reaction of the vehicles towards the light. The arrows display the motion of the vehicles, with the width of the arrow indicating the speed.

(Dewdney, 1988) and even promoted as hobby projects (Cheeseman, 1988). Four of the fundamental examples are shown in Figure 2.

The operation of these vehicles is, at once, both simple and profound. Consider these vehicles to be operating on a plain with randomly placed lights. The arcs at the front of the vehicle represent lights sensors that produce a signal based on the intensity of nearby light sources. The boxes at the back represent propulsion units that drive the vehicle at a velocity proportional to the signal that the actuator receives. In between sensors and actuators are connections that may be inhibitory or excitory. These simple components provide each vehicle with behavior representative of the labeled emotions, which is readily understood from thinking out the expected reactions of each type of vehicle. For example, the *hate* vehicle in Figure 2 will detect the light more strongly with its right sensor than its left sensor. Due to the crossed excitory connections, the vehicle's left motor will travel faster than the right causing the vehicle to turn towards the light. Once facing the light the vehicle accelerates towards it in a display of aggression, smashing the light on impact.

Systems based on weighted connections form the basis of neural network research, and as such Braitenberg vehicles are readily shown to be trainable using techniques developed for artificial neural networks. Trainable structures based on Braitenberg vehicles are therefore suitable as the trainable action module of the robot. Trainability combined with simplicity makes the Braitenberg vehicle model an excellent choice for a generic, low cost action generation system.

The input to the action module defines the output of the perception module. That is, the perception module should produce a pair of outputs that indicates the presence of the desired object within symmetric sensor regions, as do the sensors of a Braitenberg vehicle. The

perception module can also be trained using artificial neural network techniques, making a complete trained system.

Perception is based on natural light vision using a CCD camera. While low cost ultrasound or infra red sensors offer solutions in laboratory environments, vision is widely applicable to real world tasks. For example, a garbage collection robot must be able to distinguish between obstacles to be avoided and garbage to be collected; a fruit picking robot must be able to distiguish between the leaf and the fruit. Trainable vision offers a generic solution to sensing across a great number of situations. Because the vision system is trained, the usual overheads associated with programming are avoided.

Vision can also offer economic benefits from a hardware perspective. A camera can act as a single sensor to replace a variety of sensors, offering stock and manufacturing cost reductions. Low resolution CCD cameras are currently available for as little as $8 per part in quantity (Eccles, 1997), and the CORGI project illustrates a versatile interface to low cost microcontrollers (Wyeth, 1997).

There is an obvious limit to the capabilities of any robot designed with this methodology. The output is a function of immediate input; there is no storage of internal state. This lack of memory limits the functionality of the robot to reactive behaviors, where action is readily determined from immediate perception. Clearly a robot with this architecture cannot be expected to learn how to store and recall landmarks, for example. For the tasks described at the start of this paper, much of the required behavior is purely reactive, and that is the level that the methodologies presented here seek to address.

The following section (Section 2) of this paper gives a brief description of the physical implementation of the robot built to investigate the methodology. Section 3 then describes the methods used to train perception networks sufficient to detect the environment and the targets within the environment. Section 4 describes techniques for training the action generation module. Action training is shown in simulation and on the robot. Descriptions are given of training the robot to perform a variety of tasks, concluding with the finding of tennis balls. Section 5 concludes with some thoughts on the limits of this methodology and comparisons to related work.

2. Physical Implementation

CORGI is controlled by two DC motors arranged on either side of the robot in wheelchair fashion. The motors are designed to move CORGI along at walking speed (0.8 ms^{-1}) with sufficient torque to accelerate at 0.1g (1 ms^{-2}). These parameters allow the robot to move around at human-like speeds. The motors do not provide feedback, and since they are not stepping motors, the possibility of conventional kinematic control is precluded. The manipulator is not used in the experiments described in this paper; it is fixed in position as a camera stand.

The electronics of the robot are centered on the Motorola MC68HC16Z1 microcontroller. The processor was chosen for its wide range of peripherals, low power requirements and small DSP engine that acts at the heart of the neural network computation. Given the reliance on up to date video for all control, a video grabber was custom designed to provide direct access to video memory from the microcontroller (Wyeth, 1997). This grabber also provides an electronic pan and zoom facility, allowing electronic foveation. The robot

also has serial and parallel communications capability, a keypad and LCD screen, a power management system and sufficient on-board power for two hours of operation.

3. Learning to See

Development of the perception module involves training artificial neural networks (ANNs) to recognize relevant featurds of the environment from a grayscale camera image. This section describes the techniques used to train CORGI to recognize the obstacles that it must avoid and the tennis balls that it must locate. Results are presented for the performance of the networks developed using these techniques.

3.1. Training data

The training data for developing the neural networks were gathered from CORGI and transmitted to a PC for storage and tagging. Using serial line communications, about 100 images per minute can be stored. The stored images need to be tagged to indicate the location of the object of interest in the image. This was achieved on the PC using a mouse to indicate the position of the object.

The tagging program provides a facility to equalize the number of training examples in each class, and to present the data in an appropriate order. Having an equal number of examples in each class ensures that the network does not become biased towards a particular category simply by repeated exposure. Ordering of data prevents the network from "forgetting" about one class as it learns about another. Of the algorithms developed for this facility, the most successful and general algorithm was one dubbed the Consistent Mean Output (CMO) algorithm. It is inspired by Pomerleau's comments on his system for bias reduction in the ALVINN system (Pomerleau, 1993). The CMO algorithm is applicable to a wider variety of problems than the techniques described by Pomerleau. The only assumption is that every output is equally likely, although the algorithm is easily modified for any known output distribution.

The algorithm cycles through the raw data, randomly choosing one in N examples to place in the training list. The value of N is chosen to reduce the likelihood of picking examples from the same class consecutively, given that data is likely to come in clusters. This parameter was set to five for the experiments described later. Patterns are chosen randomly without replacement, until the data pool of patterns becomes empty. A running average is maintained for each output of the network as each pattern is added to the training list. In between random patterns, each pattern in the raw data set is tested to see if it will bring the running averages closer to 1/M, where M is the number of outputs. If the pattern passes this test, it is included in the training list. Note that all patterns are tested even if they have already been chosen under the random selection scheme.

The CMO algorithm presents a training list that is well ordered and balanced. Significant improvements in training performance and generalization ability have been achieved using this technique. Specifically, in the obstacle detection example that follows (Section 3.5), the percentage of unseen patterns that tested to within 0.1 of the correct output increased from 59% to 89% using this technique.

3.2. Training algorithm

The network was trained with the well known back-propagation algorithm used for multi-layer perceptrons, as described in (Hertz et. al., 1991). This algorithm has various parameters that affect the ability of the network to learn the examples and to generalize to new situations. For the experiments below, sensitivity to several parameters was assessed. Convergence was found to be moderately sensitive to learning rate, with a value of 0.05 found to be most suitable. The network failed to converge without a momentum term in the weight update rule, and converged best with a momentum rate of 0.95. Initial weight size had little impact on the performance of the algorithm. Weights were randomly initialized to between 0.1 and -0.1.

Another parameter not usually adjusted for backpropagation is the sigmoid squashing constant, β. The sigmoid function is a common choice for the squashing function for networks trained with backpropagation:

$$y = \frac{1}{1 + e^{-\beta x}} \tag{1}$$

Typically, β = 1, but due to the large dimensionality of the input vector in this application, the function frequently saturated with this value. When the function is saturated, there is little derivative for the gradient descent operation of backpropagation. Accordingly, several values for β were investigated. A value of 0.125 was found to be most suitable.

The final parameter that was investigated was the number of hidden units to assign to the network. All experiments were conducted with a single layer of hidden units, with the number of units varied between 5 and 20. It was found experimentally that networks with 5 units performed worse than networks with 10 or 20 units, but that 10 and 20 unit networks were similar in performance. The subsequent experiments were all investigated with several network architectures, but in all cases the final architecture used a single layer of 10 hidden units.

3.3. Input / output format

The input format used in all experiments was a 4096 point vector representing the 64×64 pixels of the input image. Preprocessing techniques were chosen based on their prior existence, lack of complexity and wide applicability. Comparisons were made between performance with raw images, histogram equalized images, normalized images and images preprocessed with the Robert's operator (which performs edge detection). Experiments demonstrated significantly better performance for equalized images over raw images and images preprocessed with the Robert's operator. Equalized images were generally classified as well as normalized images; equalization was chosen as the faster of the two algorithms, with 20 ms per frame for equalization versus 35 ms per frame for normalization. Details of the algorithm may be found in most image processing texts (for example (Sonka, 1993).

Each network has two outputs effectively labeled *presence-on-the-left* and *presence-on-the-right*. This allows the network to capture four states: total absence of the object (0,0), presence of the object on the left (1,0), presence of the object on the right (0,1) and presence

of the object straight ahead (1,1). As such, the network interfaces well with the Braitenberg vehicles that act as controllers.

The output was derived from the (x,y) coordinates produced from the tagging operation. The x axis was broken into three bands to determine if the object was left, center or right. It was also found necessary to define a limit on the y axis for the presence of objects. Objects tagged as being present in pixel rows 55 to 64 were ignored for two reasons. First, objects beyond the distance associated with pixel row 55 were too far from the robot to be relevant to action. Second, objects at that distance were potentially quite small, only two or three pixels high, which made classification difficult.

The chosen output format was amenable to the following performance measures. Performance was measured by evaluating the number of classifications of the training set that were outside 0.1 of *allowable* values on both outputs (rejected outputs) and the number of classifications that were within 0.1 of *correct* values on both outputs (correct outputs). Allowable values were 0 and 1, with the correct values depending on the tag. Two performance indicators were generated from these numbers: a rejection rate (number rejected / total number tested) and a reliability rate (number correct / total number tested).

3.4. Implementation on CORGI

After a training session on the PC, the weights were downloaded to CORGI for real time operation. CORGI ran the feedforward operation of the network on the MC68HC16 DSP unit. Most of the time in this operation was spent in calculating the 4096 point dot product of the input vector with the unit's weight vector. The MC68HC16 supports a Repeating Multiply and Accumulate instruction, that performs the dot product at 12 cycles per point. With the CPU clocked at 16 MHz, a single hidden unit's activation was calculated in 3 ms.

To use this instruction, the data types of the weights and input points were converted to a fixed point representation. Scaling of the weights was required to trade resolution of the weight representation against possible saturation. Also the sigmoid function was implemented in a fixed point representation by using a 354 point table lookup. In practice, it was found that the output of the fixed point calculation was within 2 - 5% of the floating point calculations performed on the PC. The errors averaged out over time, and had little impact on the performance of the robot.

With all of the robot's ancillary functions operating, the robot ran two 40980 connection perception networks at 11 Hz. This performance could be significantly improved by using some of the low cost microcontrollers released lately. For example, the similarly priced SH7032 from Hitachi's Super-H series would provide an order of magnitude increase in network performance, and even wider peripheral support (Hitachi, 1996).

3.5. Obstacle detection

CORGI was trained to recognize obstacles in an enclosed office space (shown in Figure 3). The obstacles included the walls, doors and barriers that make up the perimeter of the area, providing a variety of different surfaces to detect. Lighting was provided from overhead bulbs and a curtained window. The lighting from the window caused significant

Figure 3. CORGI World – the environment in which CORGI was trained. In this environment CORGI experiences wall obstacles of three different types, the walls, the doors and the barriers. The floor tiles are speckled, non-uniform and reflect the fluorescent lights from overhead.

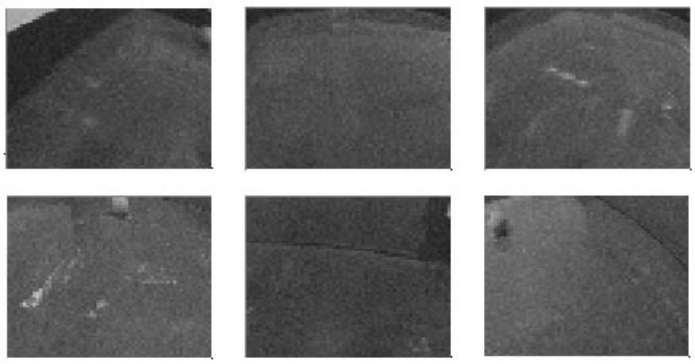

Figure 4. Some typical images from the training data gathered for obstacle detection.

variations in lighting depending on the time of day. The floor was covered with non-uniform linoleum tiles that reflected pools of light from the fluorescent lights overhead. Other features included door catches and power points. The images also contained tennis balls, for use in the subsequent experiment. Some sample images captured by the robot are shown in Figure 4.

Five thousand images were collected by driving the robot randomly around the area. The images were tagged with a single point representing the closest obstacle in the image. Data

Figure 5. Photograph of the hallway outside the office where CORGI was trained. This area was similar to the training environment, but had variations in floor color, wall color and lighting. The networks generalized well in this area.

collection took one hour, and image tagging took nearly two hours. Three thousand of these images were used in the training set, with 2000 images set aside for testing. Using the parameters described above, a network was trained on a 100 MHz Pentium based computer for 150 epochs, requiring 10.5 hours. Peak performance from the network was obtained in epoch 120, with a reliability of 89% and a rejection rate of 19% using unseen data from the training environment.

The network with peak performance was then tested on three new sets of data: sets of 100 images gathered from the same room with the blinds open, the hallway outside and a working office environment. The hall and office are shown in Figures 5 and 6 respectively.

Table 1 summarizes the results of the experiment. The change of lighting conditions and the move to the hallway had little effect on classification performance. The tests in the cluttered office showed some degradation in performance around obstacles not seen at all in the training set, particularly thin chair legs which were difficult for a human observer to detect in a 64×64 pixel image.

This experiment clearly demonstrates that an obstacle detection network trained in this manner can be a useful sensor for a robot. Not only does it function well within the environment for which it was trained, it also operates in similar previously unseen environments.

3.6. Tennis ball detection

A tennis ball was placed at random locations in the enclosed office area while data was being gathered. Of the 5000 images gathered 1845 contained taggable tennis balls. A ball was tagged only if more than 50% of the tennis ball appeared in the image. Given the low

Figure 6. The office area used in generalization experiments. The network generalized adequately in this environment, but had trouble with the legs of tables and chairs which were difficult to perceive with 64×64 resolution.

Table 1. Performance of the obstacle avoidance network beyond the training environment. The network was tested in the same room, the adjacent hallway, and a neighboring office with furniture. The degradation in performance for the neighboring office is mostly due to artifacts that are not visible with 64×64 pixel resolution.

Robot Environment	Reliability (%)	Rejected (%)
Trained environment	89	19
New lighting	84	31
Hallway	80	31
Neighboring office	64	36

Table 2. Performance of the tennis ball detection network beyond the training environment.

Ball Position	Reliability (%)	Rejected (%)
Trained environment	82	12
New lighting	80	15
Hallway	78	21
Neighboring office	45	30
Ball carried by hand	81	18

number of images containing targets, a further 4000 images were captured, bringing the total number of taggable tennis ball images to 4948. Of the 9000 images captured, 8500 were used for training with 500 set aside for testing. Total capture time came to two hours, with four hours required for the tagging of balls.

The network trained for 75 epochs and took 21 hours. Peak performance was reached in epoch 60, with a reliability of 82% and a rejection rate of 12%. The removal of balls tagged above pixel row 55 was critical to the performance of this network. With these balls present in the images, performance plummeted to 59% reliability with a 21% rejection rate. This was most likely because balls at that distance covered between 10 to 20 pixels in the image area, making them very difficult to detect.

The network generalized reasonably well for new environments. Table 2 shows the results for testing in the surrounding environments. Performance degraded more than for the obstacle network, with significant degradation in the cluttered office. The clutter in the images caused many false sightings of balls. On the other hand, the network generalized quite well for structured noise introduced into the image in the trained environment. For example, the network could still detect a ball carried by hand, and chose a tennis ball over shoes seen while testing.

4. Learning to Act

This section investigates techniques for teaching a robot to act appropriately to stimulus. The action system was based on the proposed design of Braitenberg vehicles (Braitenberg, 1984) which use a single layer of neural units to perform behavior generation and arbitration. Artificial neural network techniques to train these units were tested and then used in a series of experiments that demonstrate the capabilities of the system.

4.1. Action training algorithms

Two algorithms were chosen for action training based on their simplicity and suitability to the proposed robot architecture. The Widrow-Hoff rule (Widrow & Hoff, 1960) was investigated for training robots based on linear units, and the Perceptron Learning Algorithm (Rosenblatt, 1962) was investigated for training non-linear units. These algorithms were relevant to the proposed single layer design of the action generation and arbitration system.

The initial investigation of these systems was conducted in simulation. The simulator was designed to represent a somewhat idealized version of CORGI. While motion was calculated to be kinematically accurate, the effects of momentum and collision were ignored. Similarly, while sensing was made geometrically accurate, the reduced reliability of the sensor was ignored. This simulator provided an initial environment sufficient to judge the broad requirements for a trainable action control system. Later sections describe the implementation and experiments on CORGI.

4.1.1. Widrow-Hoff Rule The Widrow-Hoff rule (Widrow & Hoff, 1960) is based on the principle of gradient descent of error, and can generally be stated:

$$\Delta w_{ij} = \eta(\zeta_i - V_i)V_j \tag{2}$$

where η is the learning rate, ζ_i is the desired output, V_i is the actual output and V_j is the input. For the action training experiments, the input was the values from the simulated sensors, and the output was the value passed to the simulated motors. The desired behavior was obstacle avoidance.

The experiment was conducted by assigning small random values to a set of weights that fully connected the obstacle sensors to the actuators. The simulated vehicle was driven around the environment by means of key strokes. Each key stroke represented 100 ms of time. Figure 7(a) shows the path that formed the training set. This path represents 2000 sensor-actuator pairs. The learning rule was applied with a learning rate of 0.01, with the result shown in Figure 7(b). The vehicle started well, avoiding obstacles by veering away, but failed when it encountered a head-on obstacle that stimulated both sensors at once. By altering the terminating point of the training epoch a different behavior was developed. The robot avoids head-on obstacles but fails to deal with obstacles on the right (Figure 7(c)).

This learning behavior can be explained by observing the error $(\zeta - V)$ during training. Due to the fine grain nature of data collection, many like examples come in clusters. It is the nature of the Widrow-Hoff rule to minimize the error for those examples. Despite the low value given for the learning rate, the network will tend to erase the learning from previous encounters as it minimizes the error for the current situation. This single mindedness explains the behavior described above. It is feature of machine learning that is well documented, and is often described (originally by (Carpenter & Grossberg, 1987)) as the *stability-plasticity dilemma*.

This problem can be overcome in the same manner that was described for the perception module: store the training set and present it in random order. As shown in Figure 8, this solution is technically sound, but the overheads in storage and computation make this method unsuitable for an embedded design.

4.1.2. The Perceptron Learning Algorithm The Perceptron Learning Algorithm (PLA) (Rosenblatt, 1962) is designed to train threshold units. Threshold units have two valued outputs: 0 and 1. The units have an output of 0 until input activation reaches the threshold, when they have an output of 1. The algorithm was implemented as:

$$\Delta w_{ij} = \begin{cases} 2\eta\zeta_i V_j, \text{ if } \zeta_i \neq V_i; \\ 0, \text{ otherwise.} \end{cases} \tag{3}$$

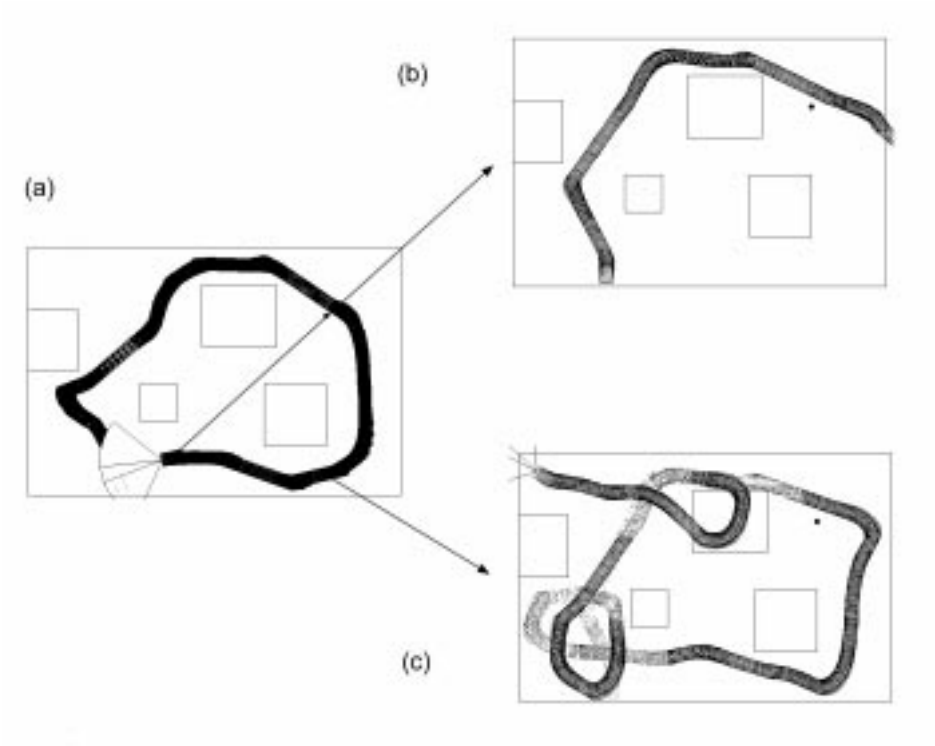

Figure 7. (a) The path for training the vehicle. (b) The result of a test run. (c) By altering the termination point for training, the behavior of the vehicle is significantly altered.

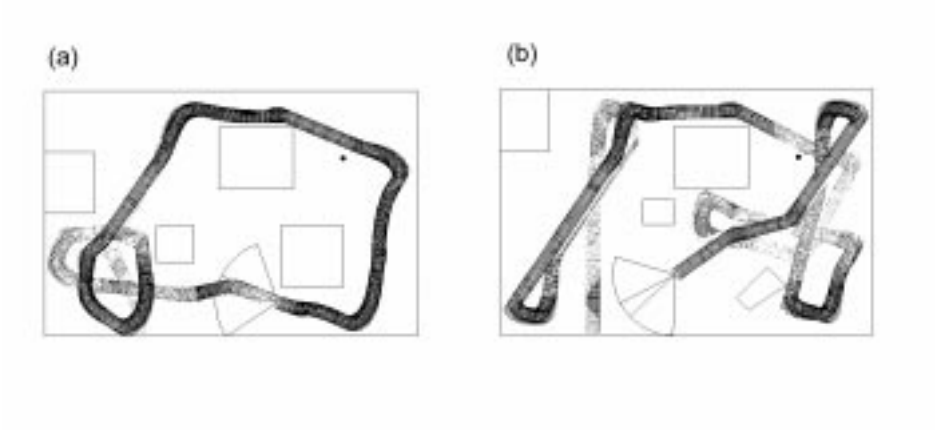

Figure 8. (a) This vehicle is also trained using the path shown in Figure 3.8(a), but with the order of presentation randomized. (b) The vehicle works in environments other than the one for which it was trained.

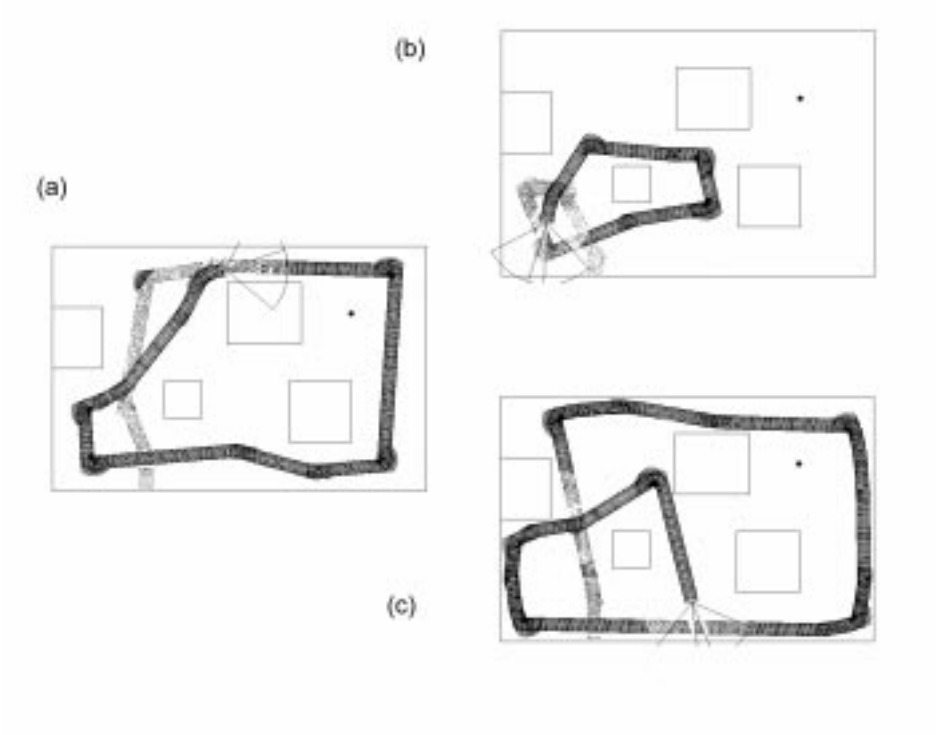

Figure 9. (a) The vehicle is trained using the perceptron learning algorithm in a single pass over the path shown. (b) Initially, the vehicle failed at the first turn, but proceeded reasonably after that. (c) By retraining the vehicle for that first turn, it was able to proceed about throughout the entire area.

with the meanings of the parameters and variables as before. The algorithm was run for 3000 sensor-actuator pairs over the path shown in Figure 9(a), using a learning rate of 0.5. When the vehicle used the learned network to generate behavior it initially had a problem at the first corner where it turned unexpectedly (Figure 9(b)). This is not unreasonable in light of the nature of the training data; the vehicle had only seen this situation once before. The behavior was "touched up" by retraining the robot through that corner. The resulting behavior was a robust obstacle avoider as shown in Figure 9(c). This behavior also worked well in previously unseen environments.

This training algorithm does not suffer from the stability-plasticity dilemma as greatly as the Widrow-Hoff rule. The PLA only updates the weights if the output is on the wrong side of the threshold, and often leaves the weights alone. This is in contrast to the Widrow-Hoff rule which will continue to adjust the weights for even small errors. The ability of the PLA to learn new behavior without corrupting older learned behavior is of great value for behavior "shaping", as (Dorrigo & Colombetti, 1995) refer to it.

4.2. Action Training Experiments

Further experiments with action training were conducted on the real robot. These experiments were all conducted using the PLA and threshold units. The perception networks described in the previous section were used as input. The outputs of the network were the motor control signals. The value sent to each motor was used to control the duty cycle of the PWM controller of the motor driver - effectively controlling the voltage supplied to each motor. This does not imply accurate velocity or position control, as was seen with the simulator.

This feature has implications for the architecture used to control the robot. If each motor is controlled by a single threshold unit, the controller only has the ability to drive a motor forward or to switch off the motor. It does not have any braking ability or reverse ability. This means that the robot can not rapidly overcome its momentum to negotiate obstacles. A possible solution is to use the threshold unit to have one state representing forward and another representing backwards. This presents problems when trying to keep the robot stationary, for instance for ball collection. A better solution is to associate two threshold units with each motor: one unit for forward and one unit for reverse. This allows for a wider variety of behavior to be learned.

The complete architecture of CORGI is shown in Figure 10. The 64×64 pixel equalized input is used as input for both the obstacle detection and tennis ball detection network. The outputs of the detection networks form the inputs of the action network. The action network behaves like a Braitenberg vehicle, using left and right sensory information to generate motor associations. The units in the action network are threshold units, with the weighted connections and unit thresholds trained using PLA. The connections from the threshold units to the motor units are fixed to produce the forward unit and backward unit described above.

4.2.1. Wall Following The robot was trained to follow right hand walls around the training area. The robot was driven around the area three times under joystick control, representing about 200 sensor-actuator pairs. Only the obstacle sensor readings were used as input for the network. The sensor-actuator pairs were presented to the learning algorithm as they were gathered, allowing for real time learning at about 8 Hz. After training, the joystick was removed and the robot left to follow the trained path. The robot followed the path without incident, performing an anti-clockwise wall following operation.

Four different training sessions produced similar results. The robot never actually collided with the wall during two hours of testing and always maintained an anti-clockwise motion. The robot remained within 200 mm of the trained path, and did not stall at any stage. The environment was modified by moving the adjustable partitions, with no noticeable degradation of performance. When the robot was taken into the adjoining hallway, performance remained the same even though the robot was not retrained.

As an experiment in fault tolerance, a bug was introduced into the motor driver code and the robot retrained. The bug (inspired by a fault in an early version of the code) would cause the left motor to go through stages of about 5-10 second duration where the motor would be driven at half the written value to the controller. The robot performed reasonably well, but it was noted that the robot would often run to the right of the trained path before

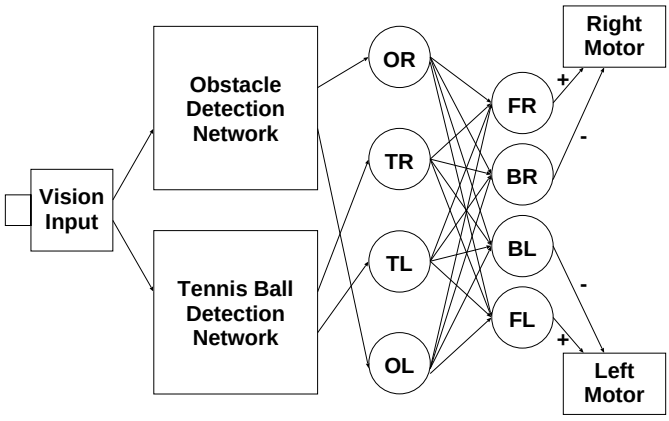

Figure 10. CORGI's complete architecture. The two trained perception networks operate on the vision input from the camera. The obstacle network produces two outputs: *obstacle-on-the-right* (OR) and *obstacle-on-the-left* (OL). Similarly the tennis ball detection network produces a right (TR) and left (TL) output. These four perception output units are totally connected to four threshold units: forward-right (FR), backward-right (BR), backward-left (BL) and forward-left (FL). The threshold units are connected with fixed excitory and inhibitory connections to the motors. The connections between the perception output units and the threshold units are trained using PLA in real time.

moving back over to the left. The trained controller was nevertheless able to deal with the problem, but in doing so introduced some bias towards turning to the right of the trained path. When the motor was lagging, the robot would move to the left for the duration of the fault.

4.2.2. A Game of Chase The robot was trained to chase tennis balls across the floor. When there was no ball present the robot would remain still, but when it saw a tennis ball it would move towards it until it collided. The collision would then push the ball away causing the robot to chase it again. This lead to the robot playing an amusing game of chase across the room.

Training was carried out in a similar fashion to before, but only the outputs from the tennis ball network were used as input for the action network. The robot learned the behavior well after only five or six games of chase, representing about one minute of actual training time, and a short period of learning to "stay" when no ball was present. In total, about 600 sensor-actuator pairs were used in the training of the behavior.

The robot's biggest problem was a lack of finesse that would sometimes cause the ball to bounce too far and become lost from view. With no view of the ball the chasing behavior halted. This is an illustration of the limitations applied by a reactive architecture. The robot could not continue to chase the ball when it lost view of it, even though the robot could conceivably tell the approximate location of the ball. It is not difficult to conceive neural structures that might help in this situation — leaky integrators on the input sensors perhaps — but it is difficult to propose a generalist architecture and accompanying training algorithm.

Despite this difficulty, the robot was able to chase a ball across the room in a variety of directions. Given the problems with keeping the ball in view, experiments were conducted carrying the ball in the hand and leading the robot around the room. The robot would follow the ball in the hand despite the noise introduced in the images by the hands and shoes that appeared in the image area.

This experiment clearly shows the advantage of a fine grained sensor-actuation control loop. The relatively high frequency of update (8 Hz when training, 11 Hz in feedforward) allows the robot to learn to chase a dynamic object such as a tennis ball.

4.2.3. Finding the Ball The most comprehensive behavior taught to CORGI was to roam the testing area avoiding the boundaries while looking for a tennis ball. When the robot came into the presence of the ball it would align itself with the ball and stop until the ball was removed. In principle, this behavior forms the basis of a wide variety of collection tasks.

The robot learned the wandering, obstacle avoidance behavior in about two minutes. Notably, the robot also learned the trainer's preference for turning left when an obstacle was approached head-on. Out of interest, a further two minutes was spent retraining the same network to reverse this tendency and turn right at head-on obstacles, which it successfully learned. Both of these networks were highly successful in that they did not once allow the robot to collide with an obstacle or get stuck in a small corner. During many demonstrations of this behavior, the robot has not once failed to avoid obstacles in the trained environment.

This behavior was then supplemented with a further two minutes of training to stop and align with tennis balls. The final resulting behavior had all of the desired features, but lacked precision in the alignment with the tennis ball. The ball would be well-aligned in a left-right sense, but the robot would stop at varying distances from the ball. To overcome this, the tennis ball perception network would require additional near-far outputs to perform control on two axes. With the addition of these outputs it appears feasible to align the robot sufficiently to perform a grasping operation with the manipulator.

This system is also moderately successful in previously unseen environments. The robot operates well in the hallway without retraining. On occasion, the robot grazed the walls as it negotiated some corners, and would sometimes drive past an apparently visible tennis ball. The robot was then tested in the neighboring office, which contains many previously unseen features. Without retraining, the robot could generally move about for three or four minutes before either colliding heavily with an obstacle, or getting trapped in a corner or under a table. Tennis ball detection became haphazard, with numerous false sightings of the ball and other balls being ignored. With five minutes of retraining, the robot typically lasted about fifteen minutes before having a fatal accident, but became notably more paranoid in the open areas of the room. The robot often turned away from regions with no apparent obstacle, and generally traveled at a lower speed. There was no noticeable improvement in tennis ball gathering ability — the poor performance is apparently a function of the perception network.

This experiment demonstrates that the action system can be taught to generate multiple behaviors and arbitrate between them. Clearly there are two behaviors at work here: an obstacle avoidance behavior, and a tennis ball detection behavior. Both behaviors act in parallel, and dominate the action of the robot at the appropriate times. The overall behav-

ior is highly reliable in the trained environment, with graceful degradation in a changing environment.

5. Discussion

The aim of the research described in this paper was to develop a generic robotic solution to the reactive domain of problems that characterize many mundane tasks well suited to being "roboticized". Tasks in this domain include garbage collection, crop picking and object retrieval from hazardous environments. Tasks of this nature require a vision sense to identify the key environmental components that form the basis of behavior. The tennis ball collection task is a simple laboratory example of this class of problem. In this section we review some limits to the solution developed and compare the solution to related work.

5.1. Limits to the Approach

The most obvious contention is that more complex environments may make training the perception system impossible. Finding the limits in this respect is difficult, as features vary from environment to environment. The chief impact of an increase in environmental complexity is an increase in the size of the training set, and a consequent increase in training time. For the problems presented here (which were not trivial) data collection and tagging took just over a typical working day, about nine hours. Training took about one and a half days of computer time. Even if these costs were increased by an order of magnitude, they are insignificant when compared with typical costs for the development of a robot vision and control system.

Clearly understood preprocessing techniques that are appropriate to the target may enhance the trainability of the system. For example, the tennis ball would have been far easier to detect if chrominance information were available, and some preprocessing applied that enhanced green.

As discussed earlier, there are limits to the applicability of the action generation system. The system can only produce reactive behaviors as it cannot store internal state. These limits have been explored by the robot, Herbert (Connell, 1991), where the apparently complex operations involved in drink can collection were all generated by a combination of reactive behaviors.

The ball finding experiment (Section 4.2.3) has shown that the system can learn multiple behaviors and arbitrate between them. However it remains unclear whether a behavior can be modified with respect to a certain sensor. For example, can this system be trained to find a tennis ball, pick it up, and subsequently avoid tennis balls as it must now return home with a full gripper? Assume that there is a sensor in the gripper that detects the presence of a gripped tennis ball. The sensor provides a cue for the required change in behavior, allowing the system to remain reactive as it does not rely on internal state. It is unclear whether the system described here could be trained to perform such a task. Modification of behavior is a useful tool that is used in Herbert to achieve its repertoire of tasks, and lack of this ability would adversely affect the usefulness of this system.

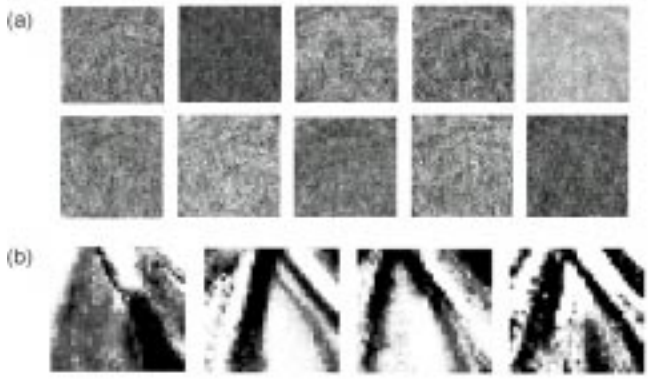

Figure 11. (a) Weight diagrams for the ten hidden units in the obstacle perception network. There is no readily apparent pattern to any of these units, and the method of obstacle detection remains unclear. (b) Weight diagrams reproduced from (Pomerleau, 1993). The edges of the roads show quite clearly in the hidden units' weights, giving a clear indication of the method used by the network.

5.2. Related Work

As outlined in the introduction, the ALVINN system (Pomerleau, 1993) bears some similarity to the perception system of CORGI. ALVINN not only perceives the current scene, it also produces the steering command. CORGI separates perception and action. This allows the difficult perception learning to be performed off-board, and more complex behavior to be trained on-board in real time. The separation greatly reduces the processing requirements for the robot. The modularity of CORGI's architecture recognizes the importance of training behavior *in situ*, but does not pay a price to attach trained behavior to trained perception.

Pomerleau illustrates in ALVINN's network that the hidden units clearly relate to certain road features. Figure 11(a) shows the weights connected to each of the ten hidden units in the obstacle detection network. There is no obvious feature that relates to obstacles apparent in these diagrams. Figure 11(b) (reproduced from (Pomerleau, 1993)) shows how clealy the edges of the road appear in ALVINN's network.

There has been a recent explosion in the number of self-learning robot behavior systems being developed. Many of these are based on reinforcement learning (for example, (Mahadeven & Connell, 1992), (Millan, 1996), (Donnart & Meyer, 1996)) and genetic algorithms ((Harvey et. al., 1994), (Floreano & Mondada, 1996), (Meeden, 1996) are a few.) with the study being centerd on unsupervized adaptation to an environment. These systems provide reward for goal completion and punishment for inappropriate behavior such as hitting obstacles. Given the lack of guiding information for the robot, these robots take a lot longer to learn appropriate behavior. The complexity of the algorithms and the long training times also means that most of these systems are developed in simulation. The type of behaviors produced by these robots are generally no more complex than obstacle

avoidance and phototaxis. The intention of these researchers is to show that robots can adapt themselves, without the requirement for a teacher.

I argue that it is simpler and more practical to teach a robot behavior, as long as that robot can generalize the behavior to new environments and behave reasonably in previously unseen situations. CORGI has shown that the simplest of neural training techniques is capable of achieving this goal. Few other researchers are investigating the use of simple supervized training techniques. Nehmzow has used supervized training techniques to train robots with many sensors and various behaviors (Nehmzow, 1995). These experiments were conducted on a range of robots from small custom made robots, to a commercially available Nomad 200 robot. These robots were equipped with typical robot sensors, ultrasound and infrared proximity detectors, tactile sensors and heavily preprocessed vision. In these experiments, where training of the control networks was performed using the Widrow-Hoff rule, the robots were able to push boxes, learn routes and clean floors.

In contrast to the experiments performed with CORGI, the learning algorithm was run with a much coarser time step, only updating every few seconds. With this arrangement, the robot could learn a typical behavior in five to ten minutes. The fine grained nature of CORGI's operation allows more rapid reaction to dynamic objects, as shown in the tennis ball chasing example. When training with such fine grain steps, the Widrow-Hoff rule used by Nehmzow becomes inappropriate, hence the need for CORGI to use the Perceptron Learning Algorithm. CORGI may be further contrasted to Nehmzow's work by the split of perception from action, with off-board training of vision, and rapid on-board training of action.

NEURO-NAV (Meng & Kak, 1993) used neural networks for real time navigation of a mobile robot, through real environments. It was designed to operate in a system of corridors and junctions, and accept commands such as "proceed to the next junction and turn left". Neural networks were used to perform hallway following and landmark detection. To look in a little detail at one of these systems, consider the hallway navigation network. It had a 64×60 pixel grayscale image as input, that was preprocessed through an edge detection module and a Hough mapping. This served to greatly emphasize the join between walls and floor along the perspective view of a corridor. A neural network with over 1500 connections was trained using backpropagation on 72 examples, which appears to be a very low number of examples for such a large network. This probably indicates that the task of the neural network was trivial and probably better suited to a single layer network. The resulting network produced 86% correct steering angles, 10% marginally incorrect steering angles and rejected the remaining 4% of the unseen test images. It takes 2 seconds to generate a steering angle from an image, using a 16 MIPS processor.

The slow refresh rate is typical of any system that uses a large amount of preprocessing. Without dedicated vision hardware, useful preprocessing of images tends to be the time performance bottleneck of a visual system. Moreover, the preprocessing performed by this system is dedicated to the particular problem of hall following. CORGI's trainable perception system provides a greater degree of flexibility and offers a massive reduction in computation time. NEURO-NAV's interface to a higher level symbolic system is of some interest as it provides a possibility for extensions to the system presented for CORGI.

5.3. Conclusions

CORGI is a visually guided robot that can be trained to perform a variety of tasks. The robot performs apparently complex vision based tasks using only a $20 microprocessor for all computation. The trainable vision system provides a general purpose sensor for a wide variety of real world problems. For many such problems, vision may be the only answer, and with falling camera prices will often be the most cost effective. Techniques have been presented in this paper that make the training of large vision networks readily achievable. The trainable action system provides robust, fault tolerant control of the robot suitable for dynamic environments. The combination of these perception and action modules across a minimalist interface provides a solution worthy of consideration for many reactive robot designs.

References

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA.

Carpenter, G.A. & Grossberg, S. (1987) A Massively Parallel Architecture for a Self-Organising Neural Pattern Recogntion Machine. *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115.

Cheeseman, M. (1988) Build a Braitenberg Vehicle! *Electronics Australia*, vol. 50, no. 3, pp. 60-64.

Connell, J.H. (1990) *Minimalist Mobile Robotics : a colony-style arhitecture for an artificial creature*, Academic Press Inc.

Dewdney A.K. (1987) Braitenberg memoirs: vehicles for probing behavior roam a drak plain marked with lights. *Scientific American*, vol. 256, no. 3, March 1987.

Donnart, J-Y & Meyer, J-A (1996) Learning Reactive and Planning Rules in a Motivationally Autonomous Animat. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 26, no. 3, June 1996, pp. 381- 395.

Dorigo, M. & Colombetti, M. (1995) Robot Shaping: Developing autonomous agents through learning. *Artificial Intelligence*, vol. 71, no. 2, pp. 321-370.

Eccles, M. (ed.) (1997) New video camera has $8 price tag. *Electronics World*, vol. 104, no. 1739, December 1997, pp. 973.

Floreano, D. & Mondada, F. (1996) Evolution of Homing Navigation in a Real Mobile Robot. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 26, no. 3, June 1996.

Harvey, I., Husbands, P. & Cliff, D. (1994) Seeing the Light: Artificial Evolution, Real Vision, *From Animals to Animats: Proceedings of the Third International Conference on Simulation of Adaptive behavior*, The MIT Press.

Hertz, J.A., Palmer, R.G. & Krogh A.S. (1991). *Introduction to the theory of neural computation*, Addison-Wesley.

Hitachi (1996), SH7604 Hardware User Manual, Available at http://www.halsp.hitachi.com, Hitachi Web Site.

Mahadevan, S. & Connell, J. (1991) Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, vol. 55, Elsevier, pp. 311-365.

Meeden, L.A. (1996) An Incremental Approach to Developing Intelligent Neural Network Controllers for Robots. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 26, no. 3.

Meng, M., & Kak, A.C. (1993) Mobile Robot Navigation Using Neural Networks and Nonmetrical Environment Models, *IEEE Control Systems*, October 1993, pp. 30-39.

Millan, J.delR. (1995) Reinforcement learning of goal-directed obstacle-avoidance strategies in an autonomous mobile robot. *Robotics and Autonomous Systems*, vol. 15, no. 3, 1995.

Nehmzow, U. (1995) Flexible control of mobile robots through autonomous competence acquisition. *Measurement and Control*, vol. 28, pp. 48-54.

Pomerleau, D.A. (1993) *Neural Network Perception for Mobile Robot Guidance*, Kluwer Academic Publishers.

Rosenblatt, F. (1962) *Principles of Neurodynamics*, Spartan.

Sonka, M. (1993) *Image processing, analysis and machine vision.* London, Chapman and Hall Computing.

Widrow, B., & Hoff, M.E. (1960) Adaptive Switching Circuits. *1960 IRE WESCON Convention Record*, part 4, pp. 96-104.

Wyeth, G. F. (1997), Active Vision for Embedded Systems, *Proc. Mechatronics and Machine Vision in Practice*, Toowoomba, Australia, IEEE Computer Society Press, September 1997, pp. 240-245.

Received September 1, 1997
Accepted December 30, 1997
Final Manuscript February 1, 1998