LEARNING STRATEGY FOR A SIMPLE ROBOT INSECT

R.J.Mitchell, D.A.Keating, C.Kambhampati

University of Reading, U.K.

The Department of Cybernetics has recently developed some simple robot insects which can move around an environment which they perceive through simple sensors. Suitable behaviour can be programmed into these devices by humans so that, for example, the insects can avoid obstacles. This paper describes these insects and a novel method which has been developed which allows the insects to learn their own behaviour.

1. INTRODUCTION

There is much interest in the development of intelligent machines which can learn from their environment. Various machines have been developed which have many sensors, sometimes including a video camera, and so a great deal of computing power is required to process the information coming in to the machine. More processing is then required to determine suitable action in response to this information.

Researchers in the Department of Cybernetics at the University of Reading believe that it is best to start with much simpler systems. Also, we believe that there is much to learn from the behaviour of simple organisms like insects. Therefore, a number of simple robotic `insects' have been developed which are small and which can operate rapidly. The first systems built have few sensors which the insects use to determine a limited (though not trivial) behaviour. However, the insects were designed so that extra sensors could be added to allow more complex behaviour. The next section describes the insects; this is followed by a description of how their actions are programmed by humans; the final section describes how the insects learn their own behaviour.

2. MOBILE INSECTS

So far, eight simple mobile robotic `insects' and two larger devices have been produced, and the next generation insect has been designed whose sensors have been improved. These insects have been used in various teaching and research projects.

Simple mobile insects

The simple insects have two ultrasonic sensors, which enable the insect to detect how far the nearest object is in front of a sensor, and two motors, each of which can be set to move forwards or backwards at a given speed. Figure 1 shows a block diagram of the insect.

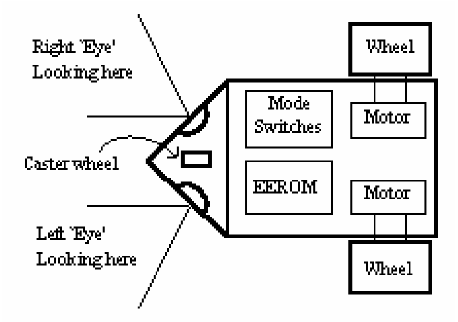

Figure 1 Block Diagram of Insect

The actions of the insect are determined using a look-up table, which is stored in an EEROM. The data from the ultrasonic sensors are passed to the address lines of the EEROM, and the value at the addressed location specifies the speed and direction of each motor. In fact, the insects can be programmed in various modes of operation which are selected by switches. The switch settings provide extra address lines to the EEROM.

Figure 2 shows a block diagram of the information paths in the insect. The ultrasonic sensors are controlled by a programmable logic array, PLA, and associated analog electronics. This PLA first causes an ultrasonic signal to be emitted by the transmitters for both eyes and then it waits for a signal to be reflected back from an obstacle to either receiver. The time taken before the reflection comes back (assuming one does return) indicates the distance of the obstacle away from the nearest eye. The time taken is determined by the PLA. The PLA produces a signal indicating the eye which is closest to any obstacle and the distance of the obstacle from that eye.

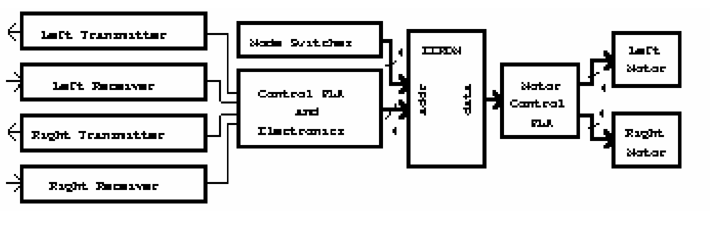

Figure 2. Information Paths in Insect

These values, together with the values set by the mode switches, are passed to the address lines of the EEROM and specify a location in a look-up table. The look-up table contains data specifying the velocity of the motors for each wheel for the different modes and sensor data. The data at the specified address in the EEROM are passed to a second PLA which converts the information to PWM signals which are passed to MOSFET bridge circuits which drive the motors.

The insect therefore moves in a manner determined by the data from its sensors, and the behaviour specified by the values in the look-up table.

Data format

Four bits of information are generated by the PLA controlling the ultrasonic eyes. One bit indicates the eye which is closest to the nearest obstacle. The remaining three bits indicate the distance of the object from the eye, giving a value in the range 0..7. If the range is 7, this indicates that no object is visible, or that any object is too far away. If the range is less than 4, then there is an obstacle which is very close. This happens because the receiver cannot start to look for a return pulse too early, or it will pick up the transmit pulse. (Means for acquiring a better value for the range are described below).

Four bits of the data from the specified address in the EEROM are passed to each motor. One bit of the four specifies the direction of the particular motor, the other three bits specify a speed value. The insect can thus, for example, go forward or backwards at various speeds, or turn, by specifying that one motor goes forwards and the other goes backwards.

Next generation of insect

As explained above, the first version of the insect does not generate much range information and does not provide independent range information for both eyes. Therefore a second generation of insect has been designed. The requirements for this insect mean that the PLA controlling the ultrasonics is insufficient. Therefore a suitable field programmable gate array (FPGA) has been designed which controls both the ultrasonics and generates suitable PWM signals to drive the motors.

This FPGA has 500 logic gates on one device and has 84 input/output pins. This is sufficiently large to allow the insect to have four ultrasonic sensors; one for each eye, one facing straight forward, and one behind. This last sensor is useful as insects have been observed reversing into objects.

The new insects have better range information, because it is possible to start `looking' for any return pulse whilst the output signal is being transmitted. When the signal is being transmitted, the receivers detect the output signal and any return signal combined, therefore a return signal is present when the received value is larger than that received from the output signal alone. Thus a returning echo is detected by comparing the received signal with a variable threshold whose shape is like that of the envelope of the transmitted signal, as shown in figure 3. The envelope of the transmitted signal is shown, as well as the shape of the threshold signal which is compared against any received signal.

Another change in these new insects is that range information is determined for each eye separately, rather than just indicating the eye closest to an object and the distance of that object from the eye. This requires that each eye should be tested at different times, otherwise the return signal reaching one eye would be detected by the next eye a short time later. Thus, a signal is transmitted from the first eye and the time for any reflection determined, and then the next eye is tested.

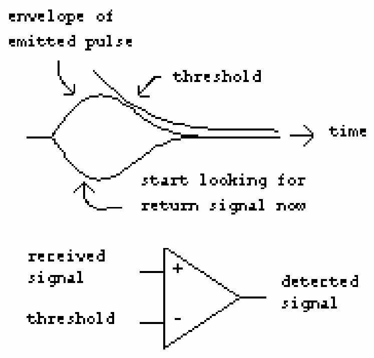
envelope of emitted pulse
threshold
time
start looking for return signal now
received signal
threshold
+
-
detected signal

Figure 3 Detection of received signal

Another improvement is the provision of a logarithmic measure of range rather than a linear one. This allows for greater accuracy when detecting the range of close objects, while still being able to detect objects further away and still using only three bits of data. The FPGA contains circuitry to achieve this.

Extra sensors

The sensors on the existing insect are ultrasonic. However, it is also possible to add extra sensors, for example thermal infra-red sensors. This, for example, could allow the insect to follow a `warm' target but avoid other obstacles, so that the insect could follow a human walking around a room but not hit table legs. This is easily achieved by connecting the output from the infra-red sensor to other address lines on the EEROM; the insect can thus switch between two behaviours depending upon the information from its sensors.

Larger insect

In addition, the Department has produced a larger version of the insect which has its own microprocessor and various extra sensors (Hole and Kelly, 1). This has been programmed to move around avoiding obstacles, but which also follows one infra-red source (a possible prey), avoids another (a possible predator), and which `docks' with a charge unit (food) when it anticipates that its battery will soon start to run down. This behaviour has been built in to the unit by a specific program.

3. PROGRAMMING THE INSECT

This section considers ways in which the actions of the insect can be programmed. As the insect does not have a microprocessor, the EEROM needs to be programmed off-line and then the EEROM inserted into the device. However, it is planned that later versions of the insect will be programmable on-line; appropriate control circuitry has been programmed into the FPGA for this purpose.

Programming the insect - by human

Determining the behaviour of the insect has been set as a laboratory experiment, in which students write four procedures determining four modes of behaviour for the insect. This is achieved by a program which generates an array, whose contents are specified by four procedures (Mode0, Mode1, Mode2 or Mode3; one for each mode) and which writes the array to a file. The data in this file are then programmed into the EEROM.

The students have to write the procedures Mode0.. Mode3, which have the form:

```
PROCEDURE Mode0 (Range : INTEGER;
                 LeftNotRight : BOOLEAN;
   VAR LeftSpeed : INTEGER;
   VAR LeftIsForward : BOOLEAN;
   VAR RightSpeed : INTEGER;
   VAR RightIsForward : BOOLEAN);
BEGIN
  << students put their code here >>
END;
```

The code written by the students determine rules for the behaviour of the insect. If the insect is to avoid obstacles, say, then the following may be appropriate.

```
 IF Range = 7 THEN BEGIN
      (* nothing visible, max speed forward *)
   LeftSpeed := 7;   LeftIsForward := TRUE;
   RightSpeed := 7;   RightIsForward := TRUE;
 END
 ELSE IF Range <= 4 THEN BEGIN
        (* something close, go backwards *)
   LeftSpeed := 7; LeftIsForward := FALSE;
   RightSpeed := 7; RightIsForward := FALSE;
 END
 ELSE IF LeftNotRight THEN BEGIN
       (* turn to the right *)
   LeftSpeed := 7; LeftIsForward := TRUE;
   RightSpeed := 7; RightIsForward := FALSE;
 END
 ELSE BEGIN  (* turn to the left *)
   LeftSpeed := 7; LeftIsForward := FALSE;
   RightSpeed := 7; RightIsForward := TRUE;
 END
```

Other behaviours can be set, for example the insect can be programmed to follow objects; if it sees nothing it stays still, otherwise it moves towards the object but stops when it gets too close.

The above has been successfully used in laboratory classes. Students have produced various strategies to enable the insect to avoid obstacles, with varying degrees of success, but also with much ingenuity; there are many solutions to the problem. The insects have also been programmed to follow objects, and six insects have `waddled' in line, just like ducklings following their mother. `Puppy' mode has also been programmed, where the insect approaches the object, but backs away when it is too close to the object.

Programming the insect - by neural network

If a learning scheme is to program the insect, then a suitable system must be trained and the result placed in the EEROM. Various methods could be used here. First the insect could be trained using a simulation and the results used in place of the Mode procedures above.

This would be appropriate if a neural network was used. For example, a multi-layer perceptron (MLP) could be trained using the back-propagation algorithm. The inputs to this network could be the Range and LeftNotRight parameters passed to the Mode procedure and its outputs would be the variable parameters passed to the procedure. The trained network would then be called from the Mode procedure when the data file was generated.

There are problems with this, the main one being the training of the network. The back-propagation algorithm requires that the outputs of the network are known for each input combination, and the differences between the calculated outputs and the known outputs produce the error signals which are propagated back through the network. Although suitable outputs of the network are known (they have been programmed into the Mode procedures) which could be used to generate the error to be back-propagated, it was felt that the system should be able to generate its own values.

One solution to this problem is to use a technique like that described by Nguyen and Widrow(2) for their `Truck Backer-upper' where the system learns to reverse a truck and trailer to a specific point on a wall. This has two networks, one which was first trained to model the truck and trailer, and another which was then trained to control them. When training the controller, the truck was allowed to reverse until it hit the wall (in a simulation!), and the distance between the position of impact and the desired position generated an error. This error was then propagated through the first network so as to generate the errors at the output of the controller which were then used to train the controller. This, however, is a complicated solution to the problem. Also, the back-propagation algorithm has many defects, not the least being the long time it takes to learn.

An alternative approach is to use a neural network which uses an unsupervised learning algorithm, such as a Kohonen network. Davidov and Smele (3), two students in the Cybernetics Department, applied such a network to their models of the insect. This successfully identified different circumstances where suitable strategies were required. For example, when the insect is to avoid obstacles, the insect should have one action when there is an obstacle to the left and a different action when there is an obstacle to the right. Suitable actions were then applied in these circumstances and the insects successfully avoided obstacles.

The other major work done in the Department which is relevant to this problem was done by Ball (4). This produced a hybrid classifier system using Kohonen networks and genetic algorithms with the aim of choosing suitable activities to achieve a desired goal. This system, however, is far too complex for the insects.

Programming the insect - the requirements

One problem with the above methods is that a relatively powerful processor is needed, whereas the actual insects have no microprocessor. Therefore it was felt that a simpler, less processor-intensive strategy was required which could be applied directly to the insect. This would allow the existing insect to be enhanced by an extra FPGA, or a simple single-chip microprocessor, which directed the learning strategy. In this way, the concept of a simple insect would be maintained; a single organism would start off having no in-built behaviour, but would subsequently move around its environment learning suitable actions.

It should be noted that it is not the aim of the insect to learn the position of each obstacle in its environment and hence to devise a path to move around the obstacles. Rather, the aim is for the insect to learn more general rules which allow it to investigate any environment without bumping into obstacles; the instincts of the insect are to keep moving and avoid obstacles, and the insect learns the actions needed to achieve these. When extra sensors are added to the insect, like the provision of `food', the instincts will also include the desire to find the food and the insect will learn the appropriate actions.

4. LEARNING STRATEGY

The strategy chosen to achieve these aims is based on a fuzzy automaton algorithm, though it has been modified suitably. The basic idea, which is described in Narendra and Thathachar (5), and which has been used by Oomen et al (6) and Tsoularis et al (7), is as follows.

There are m possible actions, a1..am, associated with which are probabilities of taking these actions, p1..pm. The action which has the largest probability is then used and its performance is then evaluated as a success or a failure. As a result, the probabilities are changed according to the rules given below, where a is a value between 0 and 1, n is the number of the action with the highest probability, and j is all values 1..m except for n.

```
IF action successful THEN
  pn = pn + a (1 - pn)
  pj = (1 - a) pj
ELSE
  pn = (1 - a) pn
  pj = a / (m-1) + (1 - a) pj
```

If an action is successful, its probability is increased and that of the others is decreased; but if the action is unsuccessful its probability is decreased and the others adjusted (if pj < 1/m-1, pj is increased, otherwise it is decreased).

The above is not directly appropriate to the insect, as different strategies are required depending upon the position of obstacles relative to the insect. Therefore it was decided that one automaton would be used for each of the following five circumstances;

Case 1: no obstacle in front of an eye

Case 2: an obstacle relatively near the left eye

Case 3: an obstacle relatively near the right eye

Case 4: an obstacle in front of the left eye

Case 5: an obstacle in front of the right eye

These cases correspond to the circumstances identified by the Kohonen network. The final system, therefore, is hierarchical; simple range information is used to choose one of five automata, the chosen one being used to select a suitable action and to learn from the results of that action.

Nine possible actions were chosen, where each motor could go forwards, stop, or backwards (each motor is given a velocity of 4, 0 or -4); thus both motors could go forwards, one forward and one stopped, one forward and one backwards, etc.

The next problem was to decide whether a particular strategy was successful. This required careful thought. The aim was to produce simple `common sense' ways of evaluating the performance of an action, but ways which did not explicitly tell the insect what to do. It was decided that each action should be given an overall rating, here called a, which is used to update the probabilities, and which could be positive or negative. In general terms the rules to set a were as follows.

If no obstacle was visible then speed is important, the faster forward the higher the value of a. If an obstacle was very close to the insect, a is high if the action caused the insect to move away from the obstacle. If an obstacle is relatively close, then both factors are taken into consideration.

These rules were encoded so as to generate a suitable a value, which could be positive or negative, with which the probabilities are updated. The algorithm to do this which was described above has a fixed positive a value; this had to be changed to allow negative values.

Also, in the above rules, during an unsuccessful action, the probabilities of other rules could be reduced, which was felt to be wrong.

Therefore the following rules were used to update the probabilities, for a given a rating:

```
IF a >= 0 THEN (* if successful *)
  pn = pn + a (1 - pn)
  pj = pj (1 - a)
ELSE
  pn = pn - a
  pj = pj + a / (m-1)
```

One final change to the strategy is that the action chosen is selected randomly using a weighted roulette wheel technique, as is used in genetic algorithms (Goldberg, 8), so that the action with the highest probability is more likely to be chosen than one with a low probability. This change enables the system to escape from local maxima.

It should be noted that, as one aim of this work is to implement the strategy on an actual robot insect in either an FPGA or a single-chip microprocessor, the probabilities are implemented as 8-bit positive integers rather than as fractions.

The main adaptive loop is thus as follows.

```
Choose One Probability Set (Automaton)
Choose Action, an, of this Automaton
Move Insect for I iterations
Evaluate Action and Update Probabilities
```

The a factor is calculated by considering the speeds of the two motors, lspeed and rspeed, the ranges of obstacles from each eye, lrange and rrange, the previous values of these ranges, lrangewas and rrangewas, and the current automaton, case 1,.. case 5. The algorithm is:

```
IF case 1 THEN
  a = (lspeed + rspeed) / 4;
ELSE IF case 2 OR case 3 THEN
  a = (lspeed + rspeed) / 4 +
      (lrange - lrangewas) + (rrange - rrangewas)
ELSE
  a = (lrange - lrangewas) + (rrange - rrangewas)
```

5. EXPERIMENTS AND RESULTS

A simulation was written of an insect in a room with obstacles. This was used to test the algorithm before it could be implemented directly on the insect. The room is shown in figure 4; it has three obstacles on its walls and one in the middle. The figure shows the relative sizes of the insect, the ultrasonic beams it emits and the room.

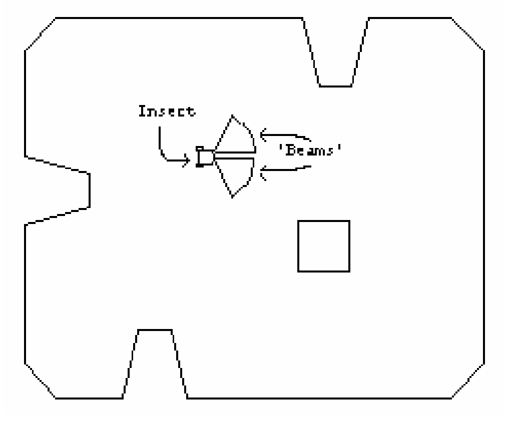

Figure 4 Room in which the inset moves

The simulation allows various factors to be set. For example, the insect can use one set of probabilities or the five sets described above; also, it is possible to specify the number of times the insect is moved before the action is evaluated (I); also the value of a (which is first calculated in the range -3..3) is then transformed either linearly to values in the range -3ar, -2ar, -ar, 0, ar, 2ar, 3ar (for some constant ar), or more emphasis is given to very good and very bad actions, the values being -9ar, -4ar, -ar, 0, ar, 4ar, 9ar. It is also possible to `freeze' the probabilities after the simulation has been run for sometime, and to fix the choice of action to the one with the highest associated probability. The most probable actions can then be encoded in a Mode procedure so as to verify that the insect has in fact learnt suitable actions.

The simulation was run many times, with the random number generator starting with different values. The insect was initially placed in the middle of an environment where there were no near obstacles. In general it very quickly learnt to move with both motors forward or one forward and one stopped, but always it eventually learnt that both motors forward was the best strategy to adopt in open space. The insects also successfully learnt to turn away from obstacles to the left or to the right by turning one motor more than another.

One interesting result from the simulation is that the insect learnt different strategies depending upon its experience. Sometimes, for example, it learnt to turn away from obstacles which were relatively near, whereas at other times the insect would learn to turn only when it got very close to an obstacle. When avoiding an obstacle on the left, the insect sometimes learnt to turn the left motor forward and the right backward, or to turn the left forward and the right off, or even the left motor off and the right backwards; sometimes two or more of these options had a high probability, so the insect would use a combination of both actions.

The insect can also move successfully when only one probability set is used, but here, for example, it has to relearn how to avoid an obstacle on the left when it has not encountered one on the left for sometime.

Once the simulation has run for sometime and the updating of the probabilities is stopped, the insect successfully moves around its environment if the five sets of probabilities are used.

Further work is required, to get the insect to learn other tasks, and this could require the addition of new sensors. The next important stage, however, is to transfer the learning strategy from the simulation to an actual insect. This should be relatively straightforward, as care has been taken to make the simulation a good model of the actual insects.

CONCLUSION

A strategy has been developed which allows simple robot insects to learn to move around an environment avoiding obstacles. This, however, is just the beginning: further work is required to enable the insects to learn other tasks like following objects; then other sensors need to be added to the insect; and finally the learning strategy has to be implemented in hardware on the actual insects.

REFERENCES

1. Hole, M. and Kelly, I., 1993, "Robot Insect 2", Internal Report, Department of Cybernetics, University of Reading.

2. Nguyen, D. and Widrow, B. 1990, Proc INNC-90, 399-407, Paris.

3. Davidov, A. and Smele, 1993, "Neural Net Control of Robot Insect", Internal Report, Department of Cybernetics, University of Reading.

4. Ball, N.R. (1991) "Cognitive Maps in Learning Classifier Systems", PhD Thesis, University of Reading, UK.

5. Narendra, K. and Thathachar, M.A.L., 1989, "Learning Automata: An Introduction", Prentice-Hall.

6. Oomen, B.J., Andrade, N. and Iyengar, S.S., 1991, Int. Jnl of Robotics Research, 10, 135-138.

7. Tsoularis, A., Kambhampati, C. and Warwick, K., 1992, Proc IEE Conf. Intelligent Systems Engineering, 13-16.

8. Goldberg, D.E., 1989, "Genetic Algorithms", Addison Wesley.