Learning Robot Behaviours by Extracting Fuzzy Rules from Demonstrated Actions.

Koren Ward¹, Alexander Zelinsky² and Phillip McKerrow¹
¹School of Information Technology and Computer Science,
The University of Wollongong, NSW, Australia, 2522
²Research School of Information Sciences and Engineering,
The Australian National University, Canberra, ACT, Australia, 0200
email: koren@uow.edu.au, Alex.Zelinsky@anu.edu.au, phillip@uow.edu.au

ABSTRACT
In this paper we describe a supervised robot learning method which enables a mobile robot to acquire the ability to follow walls and negotiate confined spaces by having these behaviours demonstrated with example actions. We achieve this by demonstrating the desired motion with a remote control while accumulating training data from the robot’s sensors and teacher’s instructions. To speed up learning and make the training data more comprehensive, additional training patterns are added to the training data by translating the demonstrated exemplars so that training data applicable to locations near the demonstrated paths are also obtained. Once sufficient training data is collected, the robot’s fuzzy rule base is generated with a fuzzy rule extraction algorithm which is tolerant to the noise and uncertainties associated with robot training data. Results of simulated and real robot experiments are provided which demonstrate the effectiveness of this approach to robot learning.

1. Introduction
Programming robots for specific behaviours can take a considerable amount of time and effort. Much of this difficulty is due to environmental inconsistencies and sensor inadequacies which make it hard for a programmer to know precisely what information will arrive from the sensors at any instant. This usually means extensive trial and error experimentation involving both simulated and real robots has to be performed in order to implement even seemingly simple behaviours with no guarantee of a satisfactory outcome being achieved. To assist in overcoming this inadequacy and reduce the time needed to program a robot we are developing methods for programming robots via human demonstration. This involves two steps: (1) demonstrating the desired behaviour with the robot via remote control while collecting training data from sensors and commands given, (2) obtaining an appropriate control function which fits the training data by means of supervised machine learning.

Previously, back propagation neural networks have been used to train mobile robots to navigate environments via supervised learning [1, 2, 3, 4] however, these methods result in only limited success due to the following reasons:
• The architecture of the network is difficult to decide.
• Long, off-line training times are often required.
• Uneven distributions of training exemplars can result in some patterns being over learnt and some not being learnt enough.
• Incremental learning may result in previously learnt patterns being forgotten unless those patterns are kept in the training data (making it very awkward and slow to keep teaching the robot new situations).
• It is difficult to unlearn incorrectly trained responses without resorting to comprehensive re-training.
• Knowledge acquired by robot remains hidden and therefore it is not possible to obtain an explanation as to why a specific response was produced at any instant.
• By not being interpretable, the acquired knowledge cannot be modified or added to by manually editing the acquired knowledge.

To improve on these shortcomings we are exploring the possibility of producing robot behaviours by extracting fuzzy rules directly from training data.

A number of techniques for generating fuzzy rules directly from data have been reported recently. Generally, these methods can be classified by the learning technique involved, the most common being fuzzy neural networks [5], genetic algorithms [6], iterative rule extraction methods [7], [8] and direct fuzzy inference techniques [9]. Although these techniques differ greatly in how fuzzy rules are arrived at, they all can be divided into two basic categories: (1) those where
the fuzzy membership functions are predefined and fixed, (2) those where rules and the membership functions are formed so as to fit the training data with the minimum required rules.

Although the use of fixed fuzzy membership functions can allow rules to be generated and accessed quickly, they generally require more rules to map the input search space and therefore have difficulty where the number of inputs is large, (e.g. 16 in the case of a typical sonar sensor ring). Despite this limitation some success at teaching a robot wall following via demonstrated examples has recently been achieved by using a Fuzzy Associative Memory (FAM) matrix [10]. This however, was achieved by reducing the sensor data comprised of 16 sonar range readings to just five inputs by coarsely grouping the sonar sensors in an asymmetric fashion so that the sensor data is reduced from 16 sonar range readings to just five inputs. Also, the resolution of range readings was further reduced by providing only a minimal number of fixed fuzzy sets to resolve each input.

Unfortunately, reducing the input search space in this way results in a number of deficiencies:

- The coarse perception makes it difficult for the robot to accurately replicate demonstrated behaviours due to an inability to sense environmental details.
- It greatly increases the likelihood of conflicting training exemplars being produced (i.e. same input vector with different output command) which can impair learning.
- The configuration of sensor groupings and structure of fuzzy membership functions has to be manually designed to suit the demonstrated behaviour resulting in increased labour and reduced versatility of the robot’s sensors.

To avoid these deficiencies we use an iterative rule extraction method involving expanding hyperboxes (as explained in Section 2) to both deduce fuzzy rules and adjust membership functions to fit the training data. This results in rules which receive all 16 sonar sensors as input and enables the robot’s perception to become automatically structured around the demonstrated behaviour. Our method differs from other iterative rule extraction methods involving hyperboxes [11], [8] in that we take sensor noise into account as well as the possibility of conflicting training patterns occurring when forming the fuzzy rules from the training data. In addition to this we make the training data more comprehensive by translating the demonstrated exemplars so that additional training patterns applicable to locations near the demonstrated paths are also obtained. Although just using the current state of the environment is unlikely to provide the robot with the ability to learn long or complex paths through environments (as was achieved in [1]), it has been demonstrated to be adequate for robots to perform simple reactive behaviours like wall following [10] and obstacle avoidance [12].

In Section 2 of this paper we describe the fuzzy rule extraction algorithm used to generate the rule base as well as the means by which the training data is expanded to make each demonstrated path more comprehensive. In Section 3 we provide results of both simulated and real robot experiments. We demonstrate the potential of this approach to robot learning by showing how effective wall following or navigation skills can be produced on a simulated robot (equipped with perfect range sensing) with just single demonstrated paths. Experiments conducted with a real robot equipped with 16 sonar sensors showed similar results could be achieved by demonstrating each behaviour with a limited number of example paths.

2. Extracting the Fuzzy Rule Base from Training Data.

The fuzzy rule base is obtained from training data by a two step process. Firstly, the input vector search space is mapped by placing hyperboxes around clusters of training data which belong to the same class (i.e. the output command). Secondly, hyperboxes are converted into fuzzy rules by representing the dimensions of the resulting hyperboxes with fuzzy membership functions.

Because the input vector has 16 dimensions corresponding to each of the robot’s sonar sensors (see Figure 1(a)), each hyperbox is given 16 upper and lower limits to specify its dimensions and location within the input vector search space as represented by the following structure:

```c
struct Hyperbox{
    int Upper[16];  // Upper bound of hyperbox dimension
    int Lower[16];  // Lower bound of hyperbox dimension
    int Command;   // Hyperbox’s output response (or class)
};
```

A command response is also provided to represent what action the robot should take when its sensor data falls within the region of the search space enclosed by the hyperbox.

In order to reduce the number of fuzzy rules needed to define the robot’s behaviours, each hyperbox is restricted to belonging to one of five trajectory commands (which can be seen in Figure 1(b)) as well a Spin-Left and Spin-Right commands. All trajectory commands cause the robot to negotiate the respective trajectory at 0.3 m/s. The Spin-Left or Spin-Right commands cause the robot to stop and rotate five degrees to the left or right respectively. Thus, for learning to succeed the robot’s rule base has to acquire sufficient rules to appropriately map the input vector search space into the seven available commands.
2.1 Mapping the Input Vector Search Space.

Many algorithms have been developed for mapping training data and deciding which mapped region of the search space belongs to what class (e.g. [7, 13, 14]). When hyperboxes are used to map the search space, deciding where in hyperspace to put each hyperbox and what size to make it becomes largely a matter of compromises. Optimally all hyperboxes should be few in number and each should enclose clusters of training data belonging to the same class. Unfortunately, with robot behaviours this objective can be difficult to achieve because when fewer hyperboxes are used to map the input search space considerable overlapping tends to occur between same and different class hyperboxes.

Overlaps between same class hyperboxes has no effect on the classification process and therefore can be tolerated or even encouraged in order to minimize the total number of hyperboxes. However, overlaps between different class hyperboxes will result in the overlapping region belonging to more than one class. To deal with this either the offending hyperboxes have to be split in order to prevent such overlaps or a conflict resolution strategy must be adopted to resolve any input vectors which happen to map to the conflicting regions. Disallowing overlaps between different class hyperboxes completely may not be a wise option in the case of robot controllers because many more hyperboxes would be needed to map all the search space. To do so could result in the robot’s rule base becoming too large which could cause response times becoming too slow.

To minimize the number of conflicting zones and keep the amount of empty space within hyperboxes to a minimum, we have devised a mapping algorithm that gradually expands hyperboxes in a controlled manner (see Figure 2). By limiting the expansion distance of hyperbox dimensions during each iteration, we effectively reduce the likelihood of different class hyperboxes from overlapping by encouraging hyperboxes to expand in all dimensions rather than becoming elongated. This occurs because elongated hyperboxes have higher probability of penetrating neighbouring hyperboxes than hyperboxes with equal sides. To further assist in reducing the final rule count, we allow a small amount of different class exemplars to become encapsulated within hyperboxes. For our experiments typical misclassification tolerances ranged from 5% for simulations up to 20% for actual robot training.

Hyperbox Mapping Algorithm:

1. Put a small box around each exemplar’s input vector;
2. Label each box with its exemplar’s class;
3. Call all of the boxes outside boxes;
4. Set ExpansionDistance to zero;
5. While ExpansionDistance < MaxDistance{
   1. Increase the ExpansionDistance;
   2. For each Outside box {
      1. For each Other box of same class {
         1. if the Other box is inside the Outside box continue;
         2. Put a Temp box around both boxes;
         3. Label the Temp box with the same class;
         4. If Temp box dimn > Outside box dimn + ExpansionDistance or
            If the Temporary box classifies the data incorrectly then
               Delete the Temporary box;
         5. else{
               Call it an outside box;
               Call the two enclosed boxes inside boxes;
            } end if
      } end for
   } end for
} end while
6. Delete all inside boxes;
7. Convert remaining outside boxes into fuzzy rules

Figure 2. Algorithm for mapping hyperbox around training data.

Although this mapping algorithm can result in a considerable amount of unclassified hyperspace being present at the end of the mapping process, this appeared to present no problem with our experiments because a default forward response was assumed if no rule fired. Also, because some overlapping between different class hyperboxes is allowed, we adopt a conflict resolution strategy based on the maximum stimulation level of fuzzy rules as the following section explains.
2.2 Deriving Fuzzy Rules from Hyperboxes.

To convert the hyperboxes into fuzzy rules we simply introduce a constant gradient to the bounds of hyperbox dimensions as shown in Figure 3. For both simulated and real robot experiments we found gradients of around 1:25 produced smoother operation of the robot. Thus, in the case where trapezoidal membership functions occur, as shown in Figure 3(b), the difference in range reading between min and max activation levels is 25cm.

![Figure 3. Converting hyperboxes dimensions into fuzzy membership functions.](image)

The resulting fuzzy rules are therefore comprised of 16 trapezoidal or triangular membership functions which can be used for classifying the input vector. We found no need to devise elaborate functions for deciding the slope of the sides of the membership functions since the rules are used only to determine the similarity (or closeness of fit) of an input vector with each rule rather than membership. The constant slope simply provides a measure of how close each input vector element is to the edge of a membership function. To classify an input vector with the fuzzy rules, we use the sum operator:

\[
y = \sum_{i=1}^{n} \mu_A(x_i)
\]

where \( n \) is the number of inputs

since this produces a far better measure of similarity than minimum or multiply operators with regards to sonar sensors. This is because sonar sensors either will or will not return a signal based on object surface characteristics and the angle of incidence between the beam and the object’s surface. Thus, an input vector may still be very similar to a particular rule even if some inputs lie outside their respective membership functions. To determine an input vector’s class the rule which fires and produces the maximum sum is accepted as the winner. A rule is considered to have fired if its resultant sum is greater than 8.0 (i.e. each input has an average membership of 0.5 or greater). If no rule fires then a forward command is issued by default as this tends to be the most commonly executed rule with most behaviours. If two or more rules fire and produce the same maximum result, the rule with an output command nearest to the previous time step’s command is chosen because this action has an increased likelihood of being consistent with the demonstrated actions. If in the case where two (or more) rules have the same maximum result and are equally near to the previous time step’s output command, then one is chosen at random.

Although the use of the sum operator may not be considered enough to classify fuzzy sets generally (in the opinion of [15]), for our experiments, it proved sufficient in providing a rough measure of similarity between the sensor data appearing at the robot’s sonar sensors and the training patterns contained within hyperboxes. Infact, experiments we conducted with the fuzzy operators min and multiply failed to produce appropriate responses with our classifier due to large amounts of unclassified input space that typically remains after training. Thus by using the sum operator, we are in fact measuring similarity rather than membership.

2.3 Extrapolating Training Data

Teaching behaviours to mobile robots via demonstration basically involves deciding what path the robot should take with respect to nearby objects and controlling the robot so that it follows that path while accumulating training exemplars for fuzzy rule extraction. Hence, by teaching the robot appropriate paths you teach the robot how to react to its environment. The difficulty with this approach is there may not only be many ways in which objects can be configured around the robot but there may be an almost infinite number of possible paths that the robot could take for the many different positions and directions the robot could find itself in. Each requiring an action to be chosen appropriate for the behaviour.

Obviously, just obtaining training exemplars from a single path would not resolve very much knowledge of how the robot should behave in other situations. On the other hand, trying to teach the robot large numbers of paths can be very time consuming and can make it hard to maintain consistent responses. Failing to maintain consistent distances to nearby walls, objects and corners when performing similar actions can result in many conflicting training patterns being produced making it uncertain what the robot should do when those input patterns occur. Furthermore, the combined training data may appear very noisy which can result in too many rules being required to accurately describe the data.

To reduce these problems and the amount of effort required to demonstrate behaviours, we produce additional training exemplars from demonstrated paths by predicting the sensor data and appropriate response of locations near the demonstrated paths. To determine appropriate responses for locations near demonstrated paths we use the path extrapolation function described
in Figure 4. This simply returns the direction the robot should head if it were positioned a given distance to the left or right of each time step’s position along the demonstrated path. So the closer the extrapolated point is to the demonstrated path the more parallel to the demonstrated path the robot should head. For our experiments we considered eight points to the left and right of the robot at 5cm intervals.

\[ a = \arctan\left( \frac{d}{d_{\text{max}}} \right) \]

where: 
- \( d \) = distance from path to extrapolated point
- \( d_{\text{max}} \) = sonar range

![Figure 4. Path extrapolation function used to make training data more comprehensive.](image)

Figure 4. Path extrapolation function used to make training data more comprehensive.

Only extrapolated points in free space are considered. To determine if an extrapolated point is in free space an occupancy grid similar to Koren and Borenstein’s histogram in motion mapping method [16] was used. To assist in predicting sonar sensor data at extrapolated positions we also store all the directions of the transmitted sonar signals which detected objects in those locations. For each extrapolated and actual robot position considered, 16 training exemplars are predicted. These exemplars represent the sensor data and appropriate command if the robot were facing each of the 16 sonar sensor directions on the robot’s sonar ring at those locations. Thus if all extrapolated points are in free space, 16 \( \times \) 17 = 272 exemplars are added to the training data at each time step. To decide the appropriate trajectory command for each extrapolated exemplar we consider the difference between the robot’s extrapolated direction and the demonstrated path’s direction as well as the distance of the extrapolated point to the path as shown in Table I. If the chosen command is on a collision course with nearby objects (based on the final contents of the occupancy grid) an appropriate spin command is used instead.

![Table I Allocating appropriate commands for extrapolated exemplars.](table)

The sensor data of each extrapolated location is determined by predicting the likely sonar range readings at those locations by using the occupancy grid. A return range reading is considered likely if it intersects an object cell at an angle which is within 8 degrees of any sonar signal direction stored in the cell. If conflicting training patterns are produced (i.e. exemplars with same input vector but different output responses) we do not attempt to resolve them by averaging or eliminating the least prevalent. Instead, we allow them to coexist in the hope that they may help influence the appropriate locations of fuzzy regions between different class hyperboxes. Finally, to reduce the size of the training data, multiple copies of same exemplars are replaced with frequency counters and any exemplars which have no return range readings are removed.

(Exemplars with no return range readings are considered redundant because a robot should always move forward if no objects are present which is the default response if no rule fires).

2.4 Teaching Robot Behaviours

By using the fuzzy rule extraction method described above with extrapolated training data, the teacher can teach a robot how to react to its environment by demonstrating example paths it should follow. Although this can provide an effective means of describing behaviours like wall following, navigating obstacles, moving along corridors or docking, as shown in Figure 5(a)-(d), it does not provide the teacher an effective means of describing object avoidance behaviour when isolated objects are involved as Figure 5(e) shows. This is because extrapolated exemplars derived from the off-side of the robot’s demonstrated avoidance path actually teach the robot to head toward the object if it happens to find itself further from the object than the demonstrated path. A possible solution to this could be to replace the path extrapolation function described in Section 2.3 with a potential field extrapolation function when obstacle avoidance is to be taught to the robot.

![Figure 5. Demonstrated and extrapolated paths for (a) wall following, (b) navigating obstacles, (c) moving along corridors, (d) docking and (e) object avoidance.](image)
3. Experimental Results

We found when perfect range sensing was used with a simulated robot, behaviours like wall following, moving along corridors and docking could be taught to the robot by demonstrating each behaviour with just one demonstrated path as shown in Figure 6. The generated behaviours were robust in that they enabled the robot to recover when placed in random locations and maintain appropriate responses when the environment was significantly changed.

<table>
<thead>
<tr>
<th>Demonstrated Path</th>
<th>Resulting Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Wall Following</td>
<td></td>
</tr>
<tr>
<td>(b) Corridor Following</td>
<td></td>
</tr>
<tr>
<td>(c) Docking</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Demonstrated paths and resulting behaviours achieved for the simulated robot. (a) Wall following, (b) moving along corridors and (c) docking in a confined space.
Teaching the robot how to negotiate a path through a cluttered environment was possible with a single path, however, if the robot regularly experienced similar input vectors at different positions among obstacles, inappropriate responses could become exhibited due to the likelihood of inconsistent commands being given for similar situations as shown in Figure 7. A potential solution to this may be achieved by using command or sensor data from previous time steps to resolve conflicts in demonstrated training data as was done in [1] with recurrent neural networks.

Figure 7. Teaching simulated robot to navigate obstacles. (a) Successfully learnt path. (b) Unsuccessfully learnt path.
Results for the real robot experiments were less favourable due to sensor noise and the highly reflective nature of ultrasonic waves. Despite this we found we were able to produce effective behaviours within structured environments with single demonstrated paths. Figure 8 shows typical traces of demonstrated paths and the consequent behaviours exhibited by the robot after rule extraction.

![Figure 8](image)

(a)  (b)  
(c)  (d)  
(e)  (f)

Figure 8. Training paths and typical behaviours exhibited by the robot after training for (a) & (b) Wall following, (c) & (d) corridor following and (e) & (f) docking.

To determine the robots capacity to learn within irregular unstructured environments we conducted wall following trials in the lab with the structured artifacts removed. Figure 9 show a typical demonstrated path and the resulting behaviour. However, the size of the rule base becomes considerably large due to the diverse range of training patterns which are generated from demonstrated paths within such environments.
Table II summarises the amount of training data and final rule count of each behaviour for the simulated and real robot experiments conducted. (The figures quoted are averages from 5 trials). Although the resulting rule bases for the real robot experiments are large, we found the rule base could be reduced considerably in size by pruning away rules which either do not or only occasionally fire when behaviours are performed over a long period of time. (The figures in brackets show the average final rules count after pruning).

<table>
<thead>
<tr>
<th>Behaviour</th>
<th>Simulation</th>
<th>Robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall following (structured environment)</td>
<td>146</td>
<td>730 (850)</td>
</tr>
<tr>
<td>Wall following (unstructured environment)</td>
<td>—</td>
<td>1351 (886)</td>
</tr>
<tr>
<td>Corridor Following</td>
<td>146</td>
<td>248 (180)</td>
</tr>
<tr>
<td>Docking</td>
<td>174</td>
<td>455 (341)</td>
</tr>
</tbody>
</table>

Table II. Number of training patterns and final rule count for each demonstrated behaviour for the simulated robot and the real robot.

4. Conclusion

Despite recent advances in robotics, programming robots to perform different behaviours still remains a considerably difficult task. Research described in this paper addresses this issue by describing a means of teaching mobile robots how to react to their environments in a desired manner. Our results show that by using the disclosed fuzzy rule extraction method with extrapolated training data, effective results can be achieved by demonstrating behaviours with example paths.

References


Koren Ward received her BCompSc degree in Computer Science from the University of Wollongong in 1995. Currently she is a PhD candidate at the University of Wollongong working on the development of rapid alternatives to existing robot behaviour learning methods. Her research interests include knowledge discovery from data, internet robotics and human-computer interfaces.

Phillip McKerrow is an associate professor in the School of Information Technology and Computer Science at the University of Wollongong. He was born in Australia in 1949. He received a BE in Electrical Engineering in 1973 from the University of New South Wales, an ME in Electrical Engineering in 1979 and a PhD in Computer Science in 1980 from the University of Wollongong. He worked in the Australian Steel industry from 1966 to 1979 first as a trainee engineer and finally as an Automation and Control Engineer. In 1979, he commenced work at the University of Wollongong as a professional officer in the department of Computer Science. In 1982, he transferred to the academic staff. Phillip has developed an intelligent robotics research laboratory and published an introductory text book in robotics. His research interests focus on ultrasonic sensing for mobile robot navigation.

Alexander Zelinsky was born in Wollongong, Australia in 1960. He worked for BHP Information Technology as a Computer Systems Engineer for 6 years before joining the University of Wollongong, Department of Computer Science as a Lecturer in 1984. Since joining Wollongong University he has been an active researcher in the robotics field, obtaining his PhD in robotics in 1991. Dr. Zelinsky spent nearly 3 years (1992-1995) working for leading robotics research laboratories in Japan as a research scientist with Prof. Shinichi Yuta at Tsukuba University, and Dr. Yasuo Kuniyoshi at the Electrotechnical Laboratory. In March 1995 he returned to the University of Wollongong, Department of Computer Science as a Senior Lecturer. In October 1996 Dr. Zelinsky joined the Australian National University, Research School of Information Science and Engineering as Head of the Robotic Systems Laboratory, where he is continuing his research into mobile robotics, co-operative multiple robots and human-robot interaction. Dr. Zelinsky is a member of the IEEE Robotics and Automation Society and is currently President of the Australian Robotics & Automation Association.