Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Fuzzy Sets and Systems 141 (2004) 107–160

FUZZY sets and systems

www.elsevier.com/locate/fss

Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online Fuzzy–Genetic system

Hani Hagras*, Victor Callaghan, Martin Colley

Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, England, UK

Abstract

In this paper we present our novel Fuzzy–Genetic techniques for the online learning and adaptation of an intelligent robotic navigator system. Such a system could be used by autonomous mobile vehicles navigating in unstructured and changing environments. In this work we focus on the online learning of the obstacle avoidance behaviour, which is an example of a behaviour that receives delayed reinforcement. We show how this behaviour can be co-ordinated with other behaviours that receive immediate reinforcement (such as goal seeking and edge following) learnt during our previous work to generate an intelligent reactive navigator that can deal with unstructured and changing outdoor environments. The system described uses a life long learning paradigm whereby it is able to dynamically adapt to new environments and update its knowledge base.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Fuzzy-Genetic system; Learning; Intelligent reactive navigator

1. Introduction

The control of autonomous intelligent mobile robotic agent operating in unstructured changing environments includes many objective difficulties. One major difficulty concerns the characteristics of the environment. In outdoor environments, such as the agricultural environment (which is one of the application benchmark for this work), the inconsistency of the terrain, the irregularity of the product and the open nature of the working environment result in complex problems of identification, sensing and control. Such problems can range from the effects of varying weather conditions on the robot sensors and traction performance, through to the need to deal with the presence of unauthorised people and animals. Another major challenge for autonomous mobile robotic agents operating in

* Corresponding author.
E-mail address: hani@essex.ac.uk (H. Hagras).

0165-0114/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0165-0114(03)00116-7

changing unstructured environments is the large amounts of uncertainty that characterises these real-world environments. On the one hand, it is not possible to have exact and complete prior knowledge of these environments: many details are usually unknown, the position of people and objects cannot be predicted a priori, passageways may be blocked, and so on. On the other hand, knowledge acquired through sensing is affected by uncertainty and imprecision. Sensor noise, the limited field of view, the conditions of observation, and the inherent difficulty of the perceptual interpretation process can all influence the quality of the sensor information.

Online learning and adaptation and life long learning using real robots are desirable traits for any robot learning algorithm operating in changing and unstructured environments where the robot explores its environment to collect sufficient samples of the necessary experience. Online learning is useful for producing intelligent machines for inaccessible environments such as underwater and flying robots or where reprogramming the robots would be difficult or expensive. In these environments it is required to perform online learning through interaction with the real environment and performing any adaptation within short time intervals. In our work we will implement online learning and show how such an approach both saves money and increases reliability by allowing the robot to automatically adapt without further programming to the changing user and environment needs it will experience throughout its lifetime.

In this work we will focus on learning the obstacle avoidance behaviour. This is an example of a behaviour that would receive delayed reinforcement when used in autonomous intelligent mobile robotic agents. These are behaviours where the reinforcement is not given during each control cycle, but at a later time. Hence the term *delayed reinforcement*. The obstacle avoidance behaviour can then be co-ordinated with other behaviours that receive immediate reinforcement (such as goal seeking and edge following) learnt in our previous work [12–16] to generate an intelligent reactive navigator. Such a navigator is then capable of operating in changing and unstructured environments. A life long learning paradigm is used, allowing the acquired experience to be accumulated in a knowledge base and reused when adapting to the new environments. Our life long learning technique gives the robot the ability to navigate in changing outdoor environments like the agricultural environment where it adapts itself to the changing environment and robot conditions by tuning the controller rules that did not perform well.

In the remaining subsections of Section 1 we explain what is meant by intelligent autonomous robotic agents and why we aim to use online learning using real robots. We then present some methods used to learn robotic agents and then we introduce reinforcement learning and show when delayed reinforcement is required. In Section 2 we introduce the idea of hierarchical fuzzy logic controllers. Section 3 introduces our hierarchical fuzzy genetic systems for online learning and we illustrate how it can be used to learn the obstacle avoidance behaviour. In Section 4 we introduce our online adaptation technique and the life long learning strategy. Section 5 presents our experimental results. In Section 6 we introduce a study of complexity of our system and comparison to other related work. Finally, conclusions are presented in Section 7.

1.1. Intelligent autonomous embedded agents and online learning

According to Kasabov [20] an intelligent agent system (IAS) should be able to learn quickly from large amounts of data. He also states that an intelligent system should be able to adapt in real time and in an on-line mode as new data is encountered. The system should also be able to accommodate, in an incremental way, any new problem solving rules as they become known. It should be memory-based, plus possess data and exemplar storage and retrieval capacities. In addition, he says that an IAS should be able to learn and improve through active interaction with the user and the environment. It should have parameters to represent short and long term memory, age, forgetting, etc. Finally he states it should be able to analyse itself in terms of behaviour, error and success. To our knowledge, no system in the field of robotic agents operating in unstructured environments such as outdoor robots had satisfied these criteria [16].

Broadly speaking this work situates itself in the recent line of research that concentrates on the realisation of artificial agents strongly coupled with the physical world. A first fundamental requirement is that agents must be grounded in that they must be able to carry on their activities in the real world, in real time according to the above definition of robotic agents. Another important point is that adaptive behaviour cannot be considered as a product of an agent considered in isolation from the world, but can only emerge from strong coupling of the agent and its environment [7]. Despite this, many embedded agent researchers regularly use simulations to test their models. However, the validity of such computer simulations to build autonomous robotic agents is often criticised and the subject of much debate. Even so computer simulations may still be very helpful in the training and testing of robotic agents models. However as Brooks [5] pointed out “it is very hard to simulate the actual dynamics of the real world”. This may imply that effort will go into solving problems that simply do not come up in real world with a physical robot and that programs which work well on simulated robots will completely fail on real robots. There are several reasons why those using computer models (simulations) to develop control systems for embedded agents operating in unstructured and changing environments may encounter problems [27]:

(a) Numerical simulations do not usually consider all the physical laws of the interaction of a real agent with its own environment, such as mass, weight, friction, inertia, etc.
(b) Physical sensors deliver uncertain values, and commands to actuators have uncertain effects, whereas simulative models often use grid-worlds and sensors that return perfect information.
(c) Physical sensors and actuators, even if apparently identical, may perform differently because of slight variations in the electronics and mechanics or because of their different positions on the robot or because of the changing weather or environmental conditions.

Even where researchers are using real robots in the real world to learn behaviours, these behaviours if learnt successfully are usually frozen within the robot. Thus if some of the robot dynamics or the environmental circumstances are changed, the robot must repeat a time-consuming learning cycle to relearn the behaviours [27]. From the above discussion it is clear that using computer simulations for developing robot controllers has significant disadvantages which are best illustrated by the fact that when transferring the learnt controllers from the simulated world to the real world these controllers will usually fail [27].

In this work we will refer to any learning carried out with user intervention and in isolation from the environment using simulation as *offline* learning. In our case learning will be done through interaction with the actual environment in a short time interval and we will call this *online* learning. Evolving the robot controllers *online* enables the learnt controller to adjust to the real noise and imprecision associated with the sensors and actuators. By doing this we can develop rules that take such defects into account, producing a realistic controller for autonomous robotic agents, grounded in the physical world that emerge from strong coupling of the robotic agent and its environment not in simulation. These robotic agents are grounded in the real world (situated, embodied and operating in real time), as adaptive behaviours cannot be considered as a product of an agent in isolation from the world, but can only emerge from strong coupling of the agent and its environment.

1.2. Methods used to develop robotic agents

The navigation of a mobile vehicle can be considered as a task of determining a collision free path that enables the vehicle to travel through an obstacle course from an initial configuration to a goal configuration.

Fuzzy sets and systems constitute one of the most fundamental and influential computational intelligence tools [38]. Given the uncertain and incomplete information an autonomous robotic agent has about the environment, fuzzy rules provide an attractive means for mapping sensor data to appropriate control actions in real time. The fuzzy logic approach seems quite promising in tackling the problem of robot navigation, as it deals with various situations without requiring to construct an analytical model of the environment. In complex unstructured environments even if a human expert can help in the specification of such a complex system, improper solutions maybe produced, since there is a high probability of neglecting some important aspects and overemphasising others. Also as the number of inputs variables increases (which is the case of mobile robots) the number of rules increase exponentially which creates much difficulty in determining large numbers of rules. In the case of navigating the mobile vehicle in complex environments, it is difficult to consistently construct the rules since there are many situations to be handled; and it is time consuming to tune the constructed rules using human experience [42]. That is why automatic design of fuzzy systems represents a very promising and challenging research area [6] where our techniques will be used.

Evolutionary algorithms constitute a class of search and optimization methods guided by the principles of natural evolution and genetics. It is the case that genetic algorithms (GA) have been successfully applied to solve a variety of difficult theoretical and practical problems by imitating the underlying processes of evolution such as selection, recombination and mutation. GA are problem independent not based on gradient information and therefore has no requirement on continuity or convexity of the solution space, caring nothing of the problem being solved, asking only that the solution be rated according how well it solves the problem [21].

The design of a fuzzy system can be formulated as a search problem in high dimensional space where each point represents a rule set, membership functions, and the corresponding system behaviour. Given some performance criteria, the performance of the system forms a hyper-surface in the space. Developing the optimal fuzzy system design is equivalent to finding the optimal location of this hyper-surface. The hyper-surface as described by Shi et al. [37] has the following characteristics:

- The hyper-surface is infinitely large since the number of possible fuzzy sets for each variable is unbounded.
- The hyper-surface is nondifferentiable since changes in the number of fuzzy sets are discrete and can have a discontinuous effect on the fuzzy system's performance.
- The hyper-surface is complex since the mapping from a fuzzy rule set to its performance is indirect and dependent on the evaluation method used.

- The hyper-surface is multi-modal since different fuzzy rule sets and/or membership functions may have similar performance.
- The hyper-surface is deceptive since similar fuzzy rule sets and membership functions may have quite different performances.

These characteristics seem to make evolutionary algorithms such as GA better candidates for searching the hyper-surface than conventional methods such as hill climbing search methods [6,37]. There is much work reported in the literature on designing fuzzy controllers using GA [2,3,4,9,17,24,26,38]. However virtually most of this work was undertaken using simulation as in conventional GA, it takes a large number of iterations to develop a good controller. Thus it is not feasible for a simple GA to learn online and adapt in real-time. The situation is worsened by the fact that most evolutionary computation methods developed so far assume that the solution space is fixed. That is, the evolution takes place within a pre-defined problem space and not in a dynamically changing and open one, thus preventing them from being used in real-time applications [20]. Hence prior to our work it was *not considered feasible for a simple GA to online learn and adapt a robotic controller* [25] in unstructured outdoor environments [16].

1.3. Reinforcement learning

Reinforcement learning (RL) is a learning strategy that can be applied to an agent that must learn its behaviour through trial and error interactions with a dynamic environment. RL is defined not by characterizing a learning problem, any algorithm that is well suited for solving the problem can be considered when using a reinforcement learning approach [8]. There are two main strategies for solving reinforcement learning problems. The first is to search the space of all possible behaviours in order to find one that performs well within the chosen environment. This is the approach normally taken by work involving genetic algorithms and genetic programming and will be the approach used in this paper. The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking particular actions in various world states [20]. This approach has been used to learn fuzzy systems for mobile robots in [1,9,47].

The problem of statistical reinforcement learning is that it learns over time by systematic trial and error in which the reinforcement learner must explicitly explore its environment. Also reinforcement learning has theoretically limited learning ability as it requires heavy learning phases and in some cases it might not be able to capture the different features of complex environments, such as an unstructured outdoor environment. There are a variety of reinforcement-learning techniques that work effectively on a variety of small problems, but very few of these techniques scale well to larger problems [19]. Of these, Q learning is the most popular and seems to be the most effective model-free algorithm for learning using delayed reinforcement, it updates the expected discounted reinforcement by taking action a in state s. It does not, however address any of the issues involved in generalising over large state and/or action spaces. In addition, it may converge quite slowly to a good policy [8]. Shwartz [35] examined the problem of adapting Q learning to an average frame. An approach which he called R learning. Using the R learning approach the agent is supposed to take actions that optimise its long-run average reward. Several researchers have found that the average reward criterion is closer to the true problem than the discounted criterion and therefore prefer to use R learning over Q learning [19]. It is difficult to use R learning to generalise to different environments [19] as *R* learning does seem to exhibit convergence problems and is quite sensitive to the choice of exploration strategy.

All the previous statistical methods has tacitly assumed that it is possible to enumerate the state and action spaces and store tables of values over them. This can only be possible in very small environments due to impractical memory requirements. It also makes inefficient use of experience [19].

The behaviours that receive delayed reinforcement in autonomous robots are the behaviours where reinforcement is not given at each control cycle, but only at a later time (*delayed reinforcement*). The robot has to be able to learn from delayed reinforcement: it may take a long sequence of actions, receiving insignificant reinforcement, then finally arrive at a state with high reinforcement. The robot must be able to learn which of its actions are desirable based on a reward that can take place arbitrarily far in the future. In some applications, reinforcement is available only when the performing system achieves a given state. For instance, this may be the case where an autonomous robotic agent is attempting to reach a moving target; it might be desired to reinforce it only when it catches the prey. The state of the performing system where its performance is evaluated is called a *reinforced state*. The action brings the performing system from a state to another, possibly different state. If the achieved state is a reinforced state, the action may directly receive reinforcement otherwise it may receive a discounted reinforcement only when one of the future actions brings the performing system to a reinforced state. This makes sense, since the achievement of a reinforced state may not depend only on the last action, but also on the state from where it has been applied, i.e. on the actions done before. Also the delayed reinforcement is suitable for behaviours where at each step the system does not perform optimally, but on average reaches an optimal state [2].

In addition, some systems need time to express certain behaviours, as is the case for the obstacle avoidance behaviour. For example, a robot blocked in a corner should maneuver to escape. It should apply the obstacle avoidance behaviour for a given period, to be able to demonstrate its ability. Only at the end of this period may it receive a reinforcement that judges its performance. If this is evaluated too early, the system will never discover how to escape, since the intermediate states are not desirable per se, but only as part of the escaping behaviour. The only possibility is to evaluate the robot's performance when it succeeds in escaping, and when it is stuck or in a collision [2].

2. Hierarchical fuzzy logic controllers (HFLC)

Most commercial fuzzy logic control (FLC) implementations feature a single layer of inferencing between two or three inputs and one or two outputs. For autonomous vehicles, however the number of inputs and outputs are usually large and the desired control behaviours are more complex. The vehicles we have being using in our indoor and outdoor experiments typically have eight sensor inputs (7 sonar inputs and a bearing sensor) and two control outputs (left and right wheel speed in case of indoor robots and steering and velocity in case of outdoor robots). If we assume that each input will be represented by three fuzzy sets and each output by four fuzzy sets, using a single layer of inferencing will lead to determining $3^8 = 6561$ rules which would be difficult, if not impossible to determine.

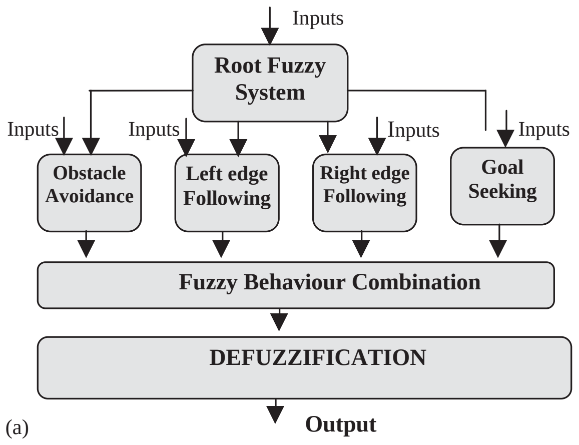
However, by using a fuzzy hierarchical approach the number of rules required can be significantly reduced. For example, the experimental system can be divided into four co-operating behaviours, obstacle avoidance, left and right edge following and goal seeking. Each individual behaviour will then only require a subset of the total number of available inputs. If, as before, the behaviours represent each input using three fuzzy sets then obstacle avoidance, using three forward facing sensors, produces $3^3 = 27$ rules. The left edge following, using two side facing sensors, produces $3^2 = 9$ rules, right edge following is the same and goal seeking, using a single location sensor, has one input (more accurately represented by seven fuzzy sets) produces 7 rules. Thus the total number of rules now required is $27 + 9 + 9 + 7 = 52$ rules which is much easier to be determined. To use such a hierarchical mechanism, a coordination scheme is required to combine these behaviours into a single action. Many proposals in the literature use an on-off subsumption system where in each situation, one behaviour is selected and is given complete control of the effectors [5]. This simple scheme may be inadequate in situations where several criteria should be taken into account.

Saffiotti [33] has suggested a fuzzy context rule combination method to perform the high level co-ordination between such behaviours. The context dependent rules are characterised by each behaviour generating *preferences* from the perspective of its goal. Each behaviour has a context of activation, representing the situations where it should be used. The preferences of all behaviours, weighted by the truth value of their contexts are fused to form a collective preference. One command is then chosen from the collective preference. Context depending blending has been used by several researchers like [3,11,33].

We will use a method similar to the methods suggested by Saffiotti [33] and Tunstel [44] to apply fuzzy logic to implement both the individual behaviour elements and the necessary arbitration (allowing both fixed and dynamic arbitration policies to be implemented) [12,13]. We achieve this by implementing each behaviour as an independent FLC with a small number of inputs and a small number of outputs which can manage simple tasks (e.g. edge-following or obstacle-avoidance). We used a FLC to implement the basic behaviours, as it excels in dealing with the imprecise and uncertain knowledge that is associated with robot's sensors and actuators.

The outputs of each fuzzy behaviour are fused according to a plan supplied by a high-level planner, which may be a person. The fusion process defines how outputs from the different behaviours are mixed together in a fuzzy way to give a coherent output. We had chosen fuzzy processes to co-ordinate the outputs of the different behaviours because it facilitates expressing partial and concurrent activation of behaviours, allowing more than one behaviour to be active to differing degrees. By doing this we can avoid the drawbacks of the binary (i.e. on-off) architectures (such as the subsumption) that allow only one behaviour to be active at a time and thus cannot deal with situations where several criteria need to be taken into account. Also, using fuzzy co-ordination processes allows a smooth transition between behaviours providing a smoother output response, which is superior to binary switching schema [33]. As mentioned earlier, using a hierarchical strategy results in fewer rules (i.e. a much simplified design problem) [33]. In addition, it allows flexible design where new behaviours can be added easily. Also, the system is capable of performing completely different tasks using the same basic behaviours by changing the co-ordination parameters. The system is totally reactive and is able to satisfy a high level objectives.

In the following design each behaviour will be a FLC using singleton fuzzifier, triangular membership functions, product inference, max-product composition and height defuzzification. The selected



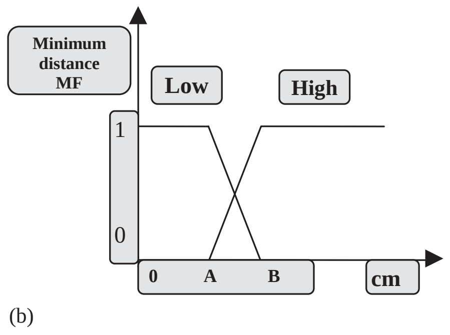


Fig. 1. (a) The behaviour co-ordination system. (b) The membership function for the co-ordination parameters.

techniques are chosen due to their computational simplicity and due to real time considerations. The standard fuzzy equation that maps the system input to output is given by

$$Y_t = \frac{\sum_{p=1}^{M} y_p \prod_{i=1}^{G} \alpha_{Aip}}{\sum_{p=1}^{M} \prod_{i=1}^{G} \alpha_{Aip}}, \tag{1}$$

where M is the total number of rules, y_p is the crisp output for each rule, $\prod \alpha_{Aip}$ is the product of the membership functions of each rule inputs and G is the number of inputs. For information about fuzzy logic please see [22,23].

In our HFLC architecture, a fuzzy operator is used to combine the preferences of different behaviours into a collective preference. Accordingly, command fusion can be decomposed into two steps: preference combination and decision making. In Fig. 1(a) each behaviour is treated as an independent fuzzy controller and then using fuzzy behaviour combination we obtain a collective fuzzy output which is then deffuzzified to obtain a final crisp output. By using fuzzy meta-rules or context rules, the proposed system enables *more flexible arbitration* policies to be achieved. These rules have the form IF context THEN behaviour [33] which means that a behaviour should be activated with a strength determined by the context (i.e. a fuzzy-logic formula). When more than one behaviour is activated, their outputs have to be fused and each behaviour output scaled by the strength of its context.

In the normal case, where fuzzy numbers are used for preferences, product-sum combination and height defuzzification, the final output equation, according to Saffiotti [33], is shown below:

$$Y_{ht} = \frac{\sum_i (mm_y^* y_t)}{\sum_i mm_y}, \tag{2}$$

where i represent the behaviours activated by context rules which can be right/left edge-following behaviour, obstacle-avoidance, goal-seeking. Y_t is the behaviour command output. These vectors have to be fused in order to produce a single vector Y_{ht} to be applied to the mobile robot. mm_y is the behaviour weight.

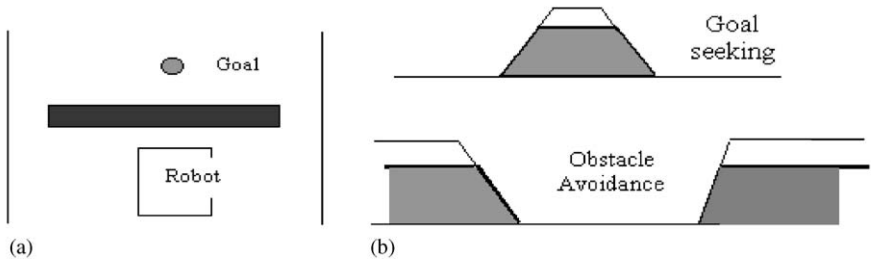


Fig. 2. (a) A robot trying to achieve a goal by avoiding an obstacle which can be equally avoided from the left or the right. (b) The Obstacle avoidance and goal seeking behaviour outputs which causes behaviour conflicts.

The behaviours implemented in this system are the minimum set of behaviours we need to demonstrate this architecture working in an outdoor environment. The behaviours are goal-seeking, obstacle-avoidance, right edge-following and left edge following.

In behaviour co-ordination there are a few parameters that must be calculated in the root fuzzy system. These parameters are the minimum distance of the front sensors which is represented by $d1$, the minimum distance of the left side sensors which is represented by $d2$, the minimum distance of the right side sensors represented by $d3$. The minimum of the fuzzy MF of $d1, d2, d3$ represented by $d4$, reflects how obstacle free the robot path is. After calculating these values, each is matched to its membership function as shown in Fig. 1(b). These fuzzy values are used as inputs to the context rules which are: *IF dl IS LOW THEN OBSTACLE AVOIDANCE*, *IF d2 IS LOW THEN LEFT WALL FOLLOWING*, *IF d3 IS LOW THEN RIGHT WALL FOLLOWING*, *IF d4 IS HIGH THEN GOAL SEEKING.*

The context rules determine which behaviour is fired, and to what degree. The final output is calculated using Eq. (2). The behaviour weights are calculated dynamically taking into account the situation of the mobile robot. For example, the obstacle avoidance behaviour weight needs to increase as the obstacle comes closer. This can be done by calculating the minimum distance of the front sensors $d1$ and then calculating the weight of the obstacle avoidance behaviour using the membership functions in Fig. 1(b). Then, by using the context rules we can determine which behaviours are active and apply Eq. (2) to obtain the final output.

In mobile robot navigation we are often faced with behaviour conflicts. For example consider the situation in which the robot objective is to navigate to a specified goal however its direct path is blocked by an obstacle which can be equally avoided from left or right as shown in Fig. 2(a). If the goal seeking and the obstacle avoidance were designed separately then the goal seeking will urge the robot to go straight ahead as the goal is aligned with the robot and the obstacle avoidance behaviour will suggest going left or right. The light lines in Fig. 2(b) shows the behaviour fuzzy output before being scaled according to its activation which is demonstrated by the thick lines and the Grey shading. If we aggregate and defuzzify these recommendations using common defuzzification techniques (e.g. centre of gravity, mean of maxima or height defuzzification) the result would be to proceed straight, thus leading to collision with the obstacle. This will occur even if a higher degree of activation was assigned for obstacle avoidance [44]. A common solution is to select a designated default alternative, or randomly select one action from the set of conflicting recommendations [44]. Some authors [46] have addressed this problem by defining different defuzzification schemes. Others put the responsibility on the behaviour designer; the rationale for this is that inconsistencies and ambiguities in a rule set should be prevented by careful design rather than corrected by arbitrary mathematical manipulation [29]. Goodridge [11] take a somehow balanced position as the occurrence of an undesirable defuzzified value is permitted, but it is reported to the higher level reasoning modules which are responsible for analysing the problem and breaking the tie.

As it is difficult to predict the interaction between the different behaviours especially when large number of behaviours are involved. We must control a priori the behaviour generation to be compatible with the combination effects, we do this not by dictating the robot what to do by assigning a default rule or random direction that can be contradictory to other behaviours. Instead we give the robot high level objectives and we allow the robot to learn rules in its coordinated behaviours that will satisfy the overall high objectives rather than the isolated objectives of the individual behaviours. For example in the problem faced in Fig. 2(a) we coordinate the goal seeking and obstacle avoidance behaviours and the rules in both behaviours are learnt online using the real robot to achieve a high level objective which is achieving a goal safely by avoiding any obstacles in its way. As will be shown later in the experiments section in Fig. 11 the robot will learn rules in the obstacle avoidance behaviour which states than when the obstacle is very near the robot has to turn sharply to the right and when it is at a medium distance from the obstacles to turn slightly right and the resulting path is smooth. Therefore, the robots had learnt rules which could be supplied by the human designer in the form of a default turning direction, but the robot had learnt this by itself through interaction with the environment. As a result we do not have to worry about the behaviour conflicts that can happen during the fuzzy combination of the behaviour outputs as the robot by trial and error and by using our online learning will generate rules in the different coordinated behaviours that will avoid such conflicts, thus relieving the designer from detailed designs of the individual behaviours that can work for some robots and fail for the others.

3. Hierarchical fuzzy genetic systems for online learning and adaptation

To achieve a high-level behaviour via the manipulation of rule-bases using a GA, it is necessary to first develop low level competence, which can then be modified and enhanced to produce an emergent high-level behaviour. This can be done at a low level by manipulating the individual behaviours, and at a higher level by manipulating combinations of the lower-level behaviours (the latter occurring once a reasonable degree of proficiency has been attained, to form a controller capable of performing a higher-level task). For this, a simple objective function is sufficient, as the controllers would already have a degree of competence for the behaviour desired. Such a solution can be achieved by implementing a hierarchical learning procedure [9].

In our hierarchical learning procedure we start learning using a set of working (but not necessarily optimum) fixed membership functions. We then commence learning general rules for each individual behaviour by relating the input sensors to the actuator outputs. In this phase the membership values are not important as the agent learns general rules such as, *if the obstacle is close then turn left.* However in order to achieve a sub-optimal solution for the individual behaviour (a subset of the large search space) we next need to find the most suitable membership functions for the newly learnt rules, as explained in [15]. After finding a sub-optimal solution for each behaviour we can combine these behaviours and learn the best co-ordination parameters that will give a “good enough” solution for the large search space to satisfy a given mission or plan, as explained in [16]. After learning the system parameters, the controller then operates in its environment where if the controller fails to maintain the desired states, the online adaptation technique modifies the poor rules in the relevant behaviours to adjust the robotic agent to differing environmental and kinematics conditions. This is termed *life long learning*, where the agent can adapt itself to any new situation and can update its knowledge about its environment.

Our learning and adaptation techniques are inspired from Nature as in biology, most scientists agree that the remarkable adaptation of some complex organisms comes as a result of the interaction of two processes, working to different time scales: evolution and life long learning. Evolution takes place at the population level and determines the basic structures of the organism. It is a slow process that works by stochastically selecting the better individuals to survive and to reproduce. Life long learning is responsible for some degree of adaptation at the individual level. It works by tuning the structures, built in accordance with the genetic information, by a process of gradual improvement of the adaptation to the surrounding environment [32]. Also condensed learning scenarios over short periods of time differ drastically from continuous learning or life long learning, as life long learning presents the agent with very different perceptual stimuli than learning over a condensed period of time [28]. We emulate the natural process by using evolution and online learning to develop a good enough controller of the robot and we use our patented Fuzzy–Genetic system (the *Associative Experience Engine*) described later to speed the slow evolution process. An online adaptation technique is then used to implement the life long learning where the robotic agent is always updating its knowledge and gaining experience and is able to adapt to the changing environment.

This hierarchical procedure results in a fast learning time for finding a solution for learning and adaptation in changing unstructured environments. Also our learning techniques produce general controllers that can be applied to different robots having the same sensor configuration and performing the same mission after applying a relatively short adaptation cycle. Thus removing the need to generate a new controller for each different vehicle.

Fig. 3 provides an architectural overview of our techniques, which we term an *Associative Experience Engine.* This forms the learning engine within the control architecture and is the subject of our British patent application 99-10539.7. The behaviours are represented as parallel fuzzy logic controllers (FLC) and form the hierarchical fuzzy control architecture presented in the previous section. Each FLC has two modifiable parameters, the *rule base* (RB) for each behaviour and the *membership functions* (MF). The behaviours receive their inputs from sensors. The output of each FLC is then fed to the actuators via the *coordinator*, which weights their effect.

The learning cycle performed is dependent upon the *Learning Focus*, which is supplied by the coordinator according to a higher-level plan. For example, if the *Learning Focus* is to learn the MF for individual behaviours, then the input membership functions of each behaviour are learnt alone.

When learning or modifying the rule bases, the learning cycle is sub-divided into local situations. This reduces the size of the model to be learnt. The accent on local models implies the possibility to learn by focusing on a small part of the search space at each step. The interaction among local models, due to the intersection of neighboring fuzzy sets causes the local learning to reflect on global performance [3]. Also in an online GA, it is desirable to achieve a high level of online performance while, at the same time reacting rapidly to any process changes requiring new actions. Hence it is required that the population size should be kept sufficiently small, so that progression towards near-convergence can be achieved within a relatively short time. This is achieved in our case by

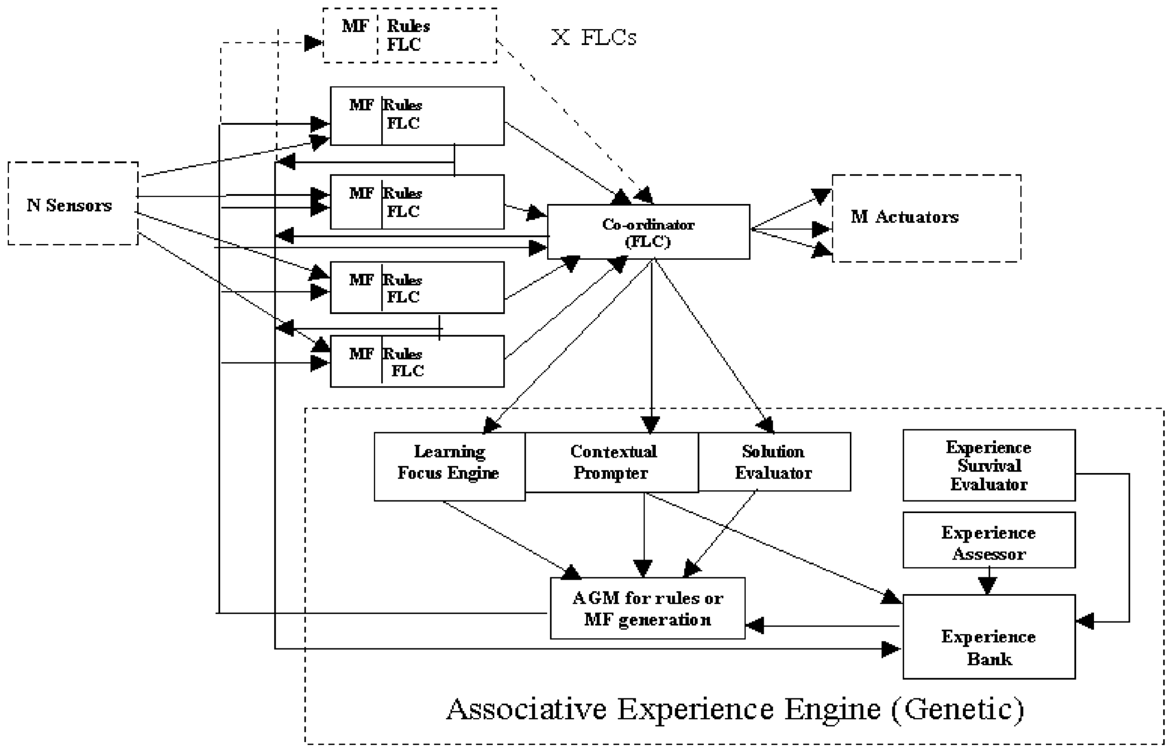


Fig. 3. Architectural overview of associative experience learning engine (British patent No 99-10539.7).

dealing with local models in the form of set of rules rather than dealing with the larger global model in the form of rule bases. Similarly the genetic operators should be used in a way that achieves high-fitness individuals in the population rapidly [25].

As GA are randomised parallel search algorithms that search from a population of points [10]. The starting population is typically randomly intialised so that the GA can proceed from an unbiased sample of the search space. However we often confront sets of similar problems. It makes little sense to start a problem solving search attempt from scratch with a random initial population when previous search attempts may have yielded useful information about the search space [42]. Instead, seeding a GA initial population with solutions to previously solved problems can provide information (a search bias) that can reduce the time taken to find a quality solution. In this case the GA does not have to waste time exploring unpromising subspaces because these starting cases provide good building blocks for solutions to the current problem [41]. As randomly initialised population has maximum capacity for exploration [41] we expect that starting from previous experiences which were solutions to similar problems to have a higher fitness than randomly generated individuals. Since the GA focuses search in the areas defined by high fitness individuals, we expect this to increase exploitation. The increased concentration or exploitation of a particular area can cause the GA to get stuck on a local optimum. Thus, we need to balance exploration of the search space versus exploitation of particular areas of the space through the crossover and the mutation probabilities [41]. As mutation serves to create random diversity in the population and as we are using small population size, mutation appears to be effective [39]. This is why it is necessary to use a high mutation rate to allow for a wider variation in the search and hence the ability to jump out of any local optimum. Thus we use a high mutation probability to introduce new genetic material without reducing the search process to a random process. It is also desirable that as the system fitness improves the mutation probability is decreased so as not to lose the genetic material that caused the improvement [25]. As the reduced crossover lowers the productivity of the GA, since there is less recombination between individuals and hence it takes a longer time to obtain a good solution, so as in [25] we set the crossover probability to 1.0 to guarantee fast convergence. In this way we speed up the search by starting from the best found point in the search space rather than starting from random and at the same time we avoid being trapped in a local optima by using the appropriate crossover and mutation probabilities.

We implement the above concepts of using the previous experiences to further reduce the search space and speeding up the search as follows. The system determines if it had encountered similar situations before by checking for stored experiences in the *Experience Bank*. There is a queue of experiences associated with each behaviour. The robot tests the different solutions from the *Experience Bank* by transferring the “remembered” experiences into the appropriate FLC. If any of these experiences show success, then they are used by the FLC and we avoid generating a new solution for our system. An *Experience Assessor* assigns each experience solution a fitness value to indicate the importance of this solution. When the *Experience Bank* becomes full it is the role of the *Experience Survival Valuer* to determine which parameters are retained and which are discarded, according to the parameter importance. If the use of past experiences did not solve the situation, we use the highest fitness experience as a starting point for a new learning cycle. We then fire an *adaptive genetic algorithm* (*AGA*) mechanism using crossover and adaptive mutation, which help to speed the search for new solutions and avoid local minimum. The AGA is constrained to produce new solutions in a certain range defined by the *Contextual Constraints* supplied by sensors and defined by the coordinator according to the *Learning Focus.* This avoids the AGA searching places where solutions are not likely to be found. By doing this we narrow the AGA search space to where we are likely to find solutions. The AGA search converges faster as it started from a good point in the search space supplied by the experience recall mechanism and it used adaptive learning parameters to avoid searching regions where solutions are not likely to be found. The new solution is then tested by the system and given a fitness by the *Solution Evaluator*, to be discussed in the next section. The AGA continues to generate new solutions until a satisfactory solution is reached.

The online learning mechanism, in addition to the fuzzy behaviours, is also organised as a hierarchy thus leading to one description of this architecture as being a “double-hierarchy”. The online learning mechanisms can be regarded as a hierarchy because there is a tiered set of actions. At the highest level a population of solutions are stored in the *Experience Bank* and tested in a queue. If one of these stored experiences leads to a solution then the search ends, if none of these stored experiences leads to a solution then each of these experiences acquires a fitness by the *Experience Assessor* depending how well each solution performed in the situation. The highest fitness experience is used as a starting position to the lower level GA that is used to generate new solutions to the current situation. This hierarchy preserves the system experience, and speeds up the genetic search by starting the genetic algorithm from the best found point in the space.

In this paper we will focus on learning the rule base for the obstacle avoidance behaviour which is an example of behaviours that receive delayed reinforcement. For more information about learning the rule bases for behaviours that receive immediate reinforcement please refer to our previous work [12,13,16]. For more information about learning the membership functions for each behaviour please refer to our work in [14]. For more information about learning the co-ordination parameters online, please refer to our work in [15].

3.1. Our patented techniques used to learn the obstacle avoidance behaviour

We will first give a brief overview of steps used and the needed parameters for the online learning of the rule base for the obstacle avoidance behaviour

(1) If the robot collides or gets close to an obstacle, the learning cycle starts. The system does not learn the whole rule base at once as in the Pittsburgh approach, but it learns a series of episodes and situations, each including learning of a small amount of rules and then relying on the intersection of neighbouring fuzzy sets causes the local learning to reflect on the global performance.

(2) The robot uses its short term memory (STM) to return to its pre failure position according to Section 3.1.1 to find the rules responsible for failure and correct them to generate a solution to the current failure. The system backs first to a distance sufficient to satisfy the minimum safe distance W from the front and X from the sides of the vehicle as will be explained in Section 3.1.1 this is called the first backing (FB). The vehicle then backs to double the FB distance which we call the second backing (SB). These two distances allow the discovery of the blamed rules for failure which if corrected could solve the situation.

(3) The system finds the two most effective rules at SB with the biggest values of S_{totSB} using Eq. (7) which are the two most effective rules contributed from the beginning of the episode till the FB position as these rules are the rules that if their actions was correct they could helped the robot early to take the right actions. The system also finds the two most effective rules at FB with the biggest values of S_{tot} using Eq. (6) as will be explained in Section 3.1.2 these rules contributed mostly through the episode and were responsible for the final failure.

(4) The system then replaces the actions of the two most effective rules at the FB and the two most effective rules at SB by rules stored in the *Experience Bank* which represents past experiences that had solved a similar situation that the robot had solved in past as will be explained in Section 3.1.3.

(5) The *Experience Bank* solutions are tested one by one in a pre specified order starting from the most successful rule clusters over the past experiences according to Section 3.1.3. If one of the experiences solves the situation then the learning ends and the rule base continues with this memorised experience with no need to generate new solutions.

(6) If all the solutions from the *Experience Bank* fail, the system allocates each experienced solution a fitness value according to how well it performed in the environment. The GA starts its search from the experienced solution with the best fitness as evaluated by the *Experience Assessor* in Section 3.1.3. Thus we do not start our learning from random but from the best point in search space, this action helps to speed up the search.

(7) In order to prevent good rules to be punished by being associated and fired by bad rules we replace first the actions of the two most effective rules at SB with the highest values of S_{totSB} as these rules will be responsible later to fire the other rules that could be good rules but could

be punished by the bad actions of these effective rules till FB and cannot recover from their bad effect. The GA population will consist of all the active rules actions from the beginning of the episode till the FB. We will use a roulette wheel selection to select the parents of the new solution. The GA will use a crossover probability of 1.0 and adaptive mutation probability given by Eq. (10) as will be explained in Section 3.1.4.1 and the rule fitness will be calculated according to Eq. (8). The system will also use the *Contextual Constraints* explained in Section 3.1.4.2 to narrow the search space.

(8) The vehicle will move to test the solution. If it solves the situation by satisfying the ending criteria given by Eq. (11) in Section 3.1.5, this is assumed as a solution and the learning cycle ends and it stores this solution in the *Experience Bank*. If the robots fails and the number of iterations is less than or equal 3 the vehicle returns to SB through FB and returns to step 7, if the number of iterations is bigger than 3 and the robot fails go to step 9.

(9) If the robot does not find a solution within a certain number of iterations, chosen empirically to be three iterations, the robot begins modifying the rules actions of the two most effective rules at FB and the two most effective rules at SB. This means that modifying the SB rules is not sufficient to produce a solution in this situation and the FB rules also need to be taken into consideration. We will call this second replacement (SR). The GA population will consist of all the active rules during the episode. The rule fitness will be calculated according to Eq. (8), we will use roulette wheel selection and a crossover probability of 1.0 and adaptive mutation probability given by Eq. (8) as will be explained in Section 3.1.4.1. The system will also use the *Contextual Constraints* explained in Section 3.1.4.2 to narrow the search space.

(10) The vehicle then starts moving with the modified rules. If the vehicle moves a distance greater than the distance needed for the ending criteria in Eq. (11) to apply to this situation, this will be considered a solution and the learning cycle ends and it stores this solution in the *Experience Bank*. If it fails and the number of iterations is less than or equal to 6 the vehicle returns to SB through FB and returns to step 9. If the robot fails number of iterations is bigger than 6 go to step 11.

(11) If the number of iterations exceeds a maximum number chosen empirically to be six without a solution being found then we decrease the situation ending criteria to half the distance. This means that this situation cannot be learnt as a single situation and must be split into two smaller situations. If distance moved is bigger then half the distance needed for the ending criteria then a solution for the situation is found and this ends the learning for the current situation and it is saved in the *Experience Bank* If not go to step 9.

Fig. 4 shows a flow chart of the steps used to learn online the obstacle avoidance behaviour.

3.1.1. Backing to identify the poorly performing rules

The movement of an autonomous vehicle can be described by a series of movement vectors (we term each movement vector a control step) $\{m(0), m(1), m(2), \ldots, m(n)\}$. The application of these vectors may cause the vehicle to get very close to or to collide with an obstacle, potentially resulting in the failure of the behaviour to act correctly. In the case of obstacle avoidance behaviour, failure is said to have occurred when the vehicle becomes too close to an obstacle as sensed by low range sonar or bumper switches. The series of the control steps that ends by colliding or getting very close to any obstacle is called an *episode*. To avoid a reoccurrence of such failures should the mobile

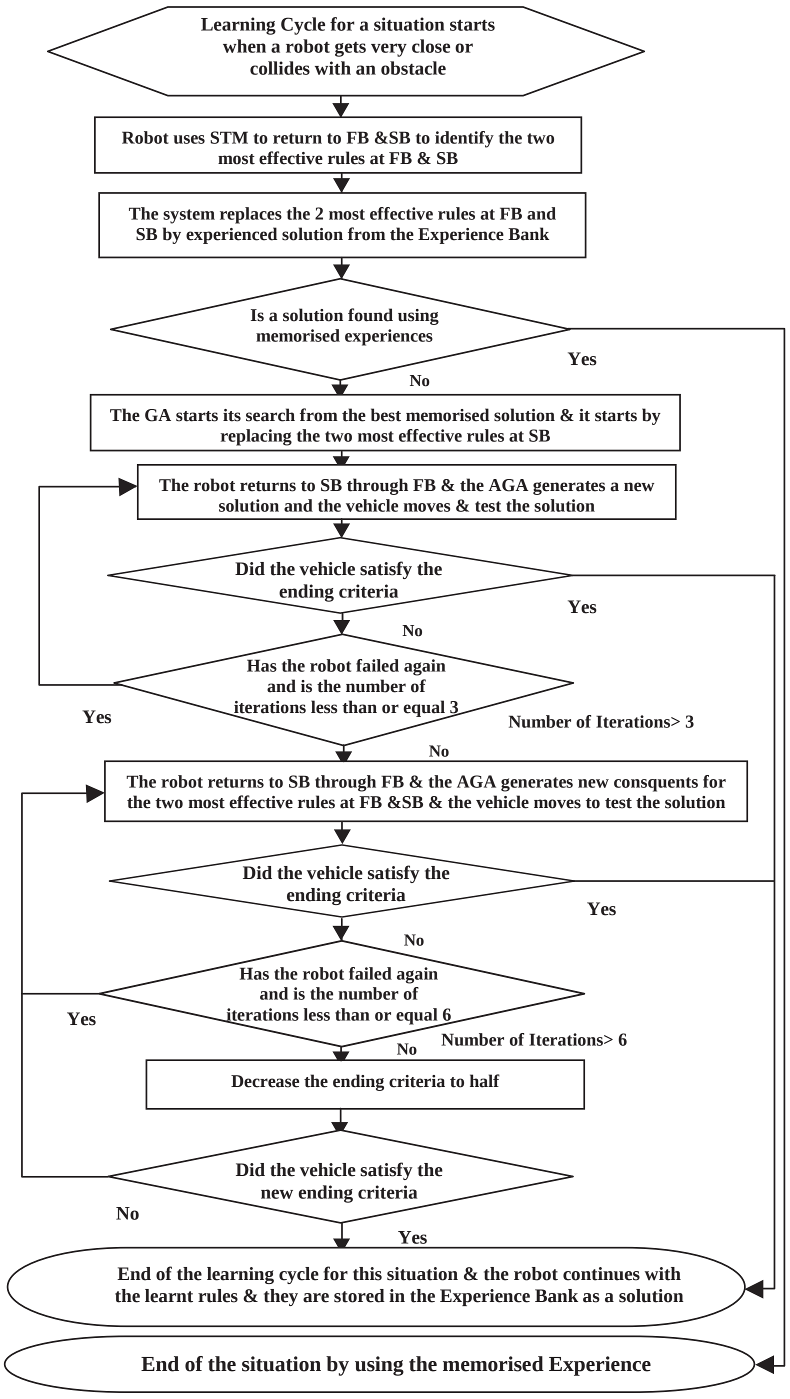

Learning Cycle for a situation starts when a robot gets very close or collides with an obstacle
Robot uses STM to return to FB &SB to identify the two most effective rules at FB & SB
The system replaces the 2 most effective rules at FB and SB by experienced solution from the Experience Bank
Is a solution found using memorised experiences
Yes
No
The GA starts its search from the best memorised solution & it starts by replacing the two most effective rules at SB
The robot returns to SB through FB & the AGA generates a new solution and the vehicle moves & test the solution
Did the vehicle satisfy the ending criteria
Yes
No
Has the robot failed again and is the number of iterations less than or equal 3
Yes
Number of Iterations> 3
No
The robot returns to SB through FB & the AGA generates new consquents for the two most effective rules at FB &SB & the vehicle moves to test the solution
Did the vehicle satisfy the ending criteria
Yes
No
Has the robot failed again and is the number of iterations less than or equal 6
Yes
Number of Iterations> 6
No
Decrease the ending criteria to half
Did the vehicle satisfy the new ending criteria
No
Yes
End of the learning cycle for this situation & the robot continues with the learnt rules & they are stored in the Experience Bank as a solution
End of the situation by using the memorised Experience

Fig. 4. A flow chart of steps used to learn online the obstacle behaviour using our Fuzzy–Genetic system.

robot navigate a similar environment, the rules that contributed mostly to generate the previous control actions must be corrected [1].

The on-line learning algorithm is used to generate new set of rules to face the future occurrences of this failure. As with classifier systems, in order to preserve the system performance the GA is only allowed to replace a subset of the rules. The worst *m* rules are replaced by *m* new rules created by the application of the GA on the population. The new rules are tested by the combined action of the performance and apportionment of credit algorithms [7]. In our case, only 4 rule consequences will be replaced (2 most effective at the first backing and 2 most effective at the second backing), these are the rules which contributed mostly to the failure situation (collision).

These rules can be found by using the vehicle's short term memory (STM). This is composed of the last *n* actions taken by the robot before the failure occurred. The vehicle can then replay the last *n* actions from the STM to move backward along the original path. The distance backed at this step will be used as the starting point for all the solutions proposed by the algorithm.

It is easy to make computations related to the vehicle dimensions so that when moving from a small to a large vehicle, the algorithm only requires parameters relating to the vehicle's dimensions to be changed. Thus making our algorithm *robot independent*. Other relevant features which discriminate among robots are manoeuvrability, speed, mass, etc. However we will only consider the robot dimension to keep computations simple and we will leave to the learning and adaptation modules the task to learn the controllers for different robots of different masses, speeds and manoeuvrability as will be shown in the experiments section.

The first step in avoiding an obstacle is to determine the minimum distance possible between the vehicle and the obstacle if the vehicle is to avoid hitting the obstacle when the maximum steering angle is applied to the vehicle. We define W to be the minimum safe distance between the front of the vehicle and the obstacle. This is the minimum distance at which the vehicle can apply the maximum steering angle and is still able to avoid hitting the obstacle. If we assume that when the maximum steering angle is applied the vehicle rotates about a front corner, the opposite corner would traverse an arc, as shown in Fig. 5(a). If the vehicle is operating in a confined space, such as a corridor, it must also determine that there is sufficient clearance to the left and right sides to safely perform the turn. We further define X to be the minimum safe distance between the sides of the vehicle and any obstacles. This can be satisfied by specifying the minimum distance from the left and right wall as X. Thus, if the vehicle rotates as described above, the rear corners of the vehicle will not impact with any obstacles during rotation.

The second step is to utilise the information stored in the Short Term Memory together with the safe distances described above to identify the badly performing rules. The vehicle replays the actions stored in the STM until it has backed away from the obstacle sufficiently to satisfy the minimum distances W and X. We will call this distance the first backing (FB) distance. This location represents the point where the maximum steering angle must be applied to avoid the obstacle, if the vehicle were to back further away, less steering angle could be applied to avoid the obstacle. This is similar to a driver near an end of a corner tries maximum steering to get out of this situation, while if he backed more he can easily get out of this situation. Thus to make the manoeuvring easier, the vehicle continues to back away until the distance from the obstacle is double the FB distance. We call this distance the second backing (SB) distance. The first and second backing distances represent waypoints where the vehicle should be taking action to avoid the obstacle. This technique is beneficial when encountering dead ends or when the space is too tight for the vehicle to manoeuvre.

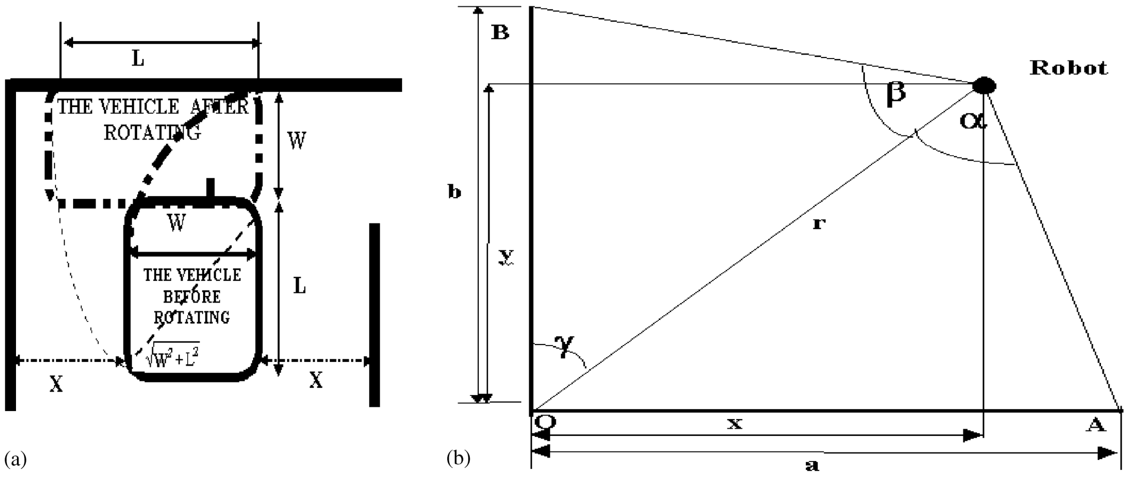


Fig. 5. (a) The vehicle turning distances. (b) The position estimation using the 3 infrared beacons.

In robotic learning using evolutionary computation, a central point is the ability of the robot to go back to its starting position to compare different solutions as the robot evolve new solutions and try them and assign fitness according the performance of each solution. So it is always necessary to start the evaluation from the same point so that all solutions are evaluated in a fair way. Also if the distance travelled before failure will be used to measure fitness then a reliable way will be needed to measure distance. In simulation and offline learning it is very easy to determine exactly, the distance the robot had moved, and always return the robot back to same position to begin assessing a new solution as the whole learning cycle is implemented using a computer simulation program. When learning in simulation, usually researchers neglect the time the robot takes to go back to its initial position to try a new solution [27]. However in real world it is difficult to precisely measure the distance travelled by the robot and to exactly return it pack to same starting point due to the noise and uncertainty in sensors and actuators.

In the lab experiments to determine this distance we have used three infrared beacons placed at right angels and at known distances **a**, **b**, and the infrared scanner sensor mounted on the robot gives bearing of the robot w.r.t. to three beacons, the system configuration is shown in Fig. 5(b). In the outdoor environment we perform this by using an electronic compass or a GPS sensors and three known positions for triangulation. The distance of the robot from a point O (beacon number zero) is given by [18]:

$$r = \frac{b\sin(\beta+\gamma)}{\sin(\beta)}, \tag{3}$$

where

γ is given by, $\gamma = \tan^{-1}((c\sin(\beta) - \cos(\alpha))/(\sin(\alpha) - c\cos(\beta)))$ and

C is given by $(b\sin(\alpha)/a\sin(\beta))$.

When the robot fails to fulfil its desired behaviour (being very near or colliding with an obstacle), the robot performs both the FB and the SB. At the end of the SB, it calculates its distance r from point O, which will be the original point for any solution evaluation. This distance is denoted by r_0 and γ_0. If the robot moves a new distance r_1, then d_{new} which is the distance moved by the robot after applying the new rule-base will be equal to

$$d_{\text{new}} = \sqrt{(r_1 \sin(\gamma_1) - r_0 \sin(\gamma_0))^2 + (r_1 \cos(\gamma_1) - r_0 \cos(\gamma_0))^2}. \tag{4}$$

In our case in order to return back to the starting position we will use a Short Term Memory (STM). This is composed of the last n actions taken by the robot before the failure occurred. Each memory entry is the actuators actions and how far did the robot travel using these actions measured by the wheel encoders. The vehicle can then replay the last n actions from the STM to move backward along the original path. When the robot finishes playing back its actions there will be an accumulative drift as a result of errors associated with the encoders. This will be corrected by using the triangulation information explained above which tells the robots its relative position relative to triangulation points (infrared beacons indoors and GPS points outdoors). We will then use a low level pilot fuzzy logic controller in which one input to the controller will be the set point which is the coordinates of the starting point, the other input will be the actual value which is the robot drifted coordinates after replaying the STM actions and the fuzzy controller output is to tune the robot actuators so that it will restart again from the original starting position. Given that we are dealing with imprecise sensors and actuators there will be a drift anyway. However what really matters is that the drift is small enough so that each time we trigger the same fuzzy rules. We are able to achieve this, as the drifts are very small not exceeding 2 cm in any direction thanks to our STM and triangulation and the pilot fuzzy logic controller that gets the robot back to the same starting position.

As mentioned previously, we cannot replace all the rules in the population, so we must choose which part of the population to replace. We choose to replace the actions of the two rules at each backing location that contributed most to the failure of the behaviour. At each backing position the rules that were used are evaluated and the contribution of each rule to the final action is determined. The greater the contribution the more the rule will be blamed for the failure and is punished by reducing its initial fitness with respect to other rules.

3.1.2. Fitness determination and credit assignment

The fitness of the system is generated by the *Solution Evaluator*. In the case of obstacle avoidance, fitness can be determined as a function of the distance moved by the vehicle [48].

In this paper, we are concerned with making the robot learn online and adapt the rules necessary for the task of avoiding obstacles. Introducing the robot to different situations, such as corridors, obstacles and walls could do this, thereby allowing the robot to discover the rules needed in each situation. The learning session consists of learning different situations or episodes. The model to be learnt is small and so is the search space. The accent on local models implies the possibility to learn by focusing at each step on small parts of the search space only, thus reducing useless interaction among partial solutions. The interaction among local models, due to the intersection of neighbouring fuzzy sets causes that local learning to reflect on global performance [3]. Moreover, the smooth transition among the different models implemented by fuzzy rules implies robustness with respect to data noise. So we can have global results coming from the combination of local models and smooth transition between close models.

In the case of the obstacle avoidance behaviour, reinforcement is available only when the performing system collides or escapes from collision after an ending criteria. The only possibility is to evaluate the robot's performance when it succeeds in escaping, or when it is collides or gets close to an obstacle. We evaluate the performance of a robot after a sequence of control steps called *episodes* ending with the robot colliding or getting very close to an obstacle. Thus the number of control is step per episode is not constant and varies from one episode to the other. At the end of each episode, the reinforcement program re-evaluates the vehicle's performance and distributes the corresponding reinforcement values to the rules that have contributed to control actions during the episode. This evaluation strategy averages the effects of the single rules, and in general has a stabilising effect [2].

In the following actions we choose not to use the bucket brigade algorithm for apportionment of credit assignment. As discussed in [19], the bucket-brigade algorithm may loose effectiveness as action sequences get longer and as we are using the FLC system we will have long chains of rules. In addition, the bucket brigade algorithm does not appear to handle partially observed environments robustly [19]. So we will only apply credit assignment to the rules that contributed during the episode, as it will be shown that modifying the most effective rules is sufficient to find a solution and there is no need for backward chaining.

Each rule p output (Y_p) contributes a variable amount to the crisp output Y_t given in Eq. (1). If there are N output variables, then we have $Y_{t1},\ Y_{t2},\ldots,Y_{tn}$ outputs. Therefore the normalised contribution of each rule p output $(Y_{p1},Y_{p2},\ldots,Y_{pn})$ to each of the outputs $Y_{t1},Y_{t2},\ldots,Y_{tn}$ can be denoted by $S_{r1},S_{r2},\ldots,S_{rn}$. Where $S_{r1},S_{r2},\ldots,S_{rn}$ are given by

$$S_{r1}=\frac{Y_{p1}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}/\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}}{Y_{t1}},\quad S_{r2}=\frac{Y_{p2}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}/\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}}{Y_{t2}},\ldots,$$

$$S_{rn}=\frac{Y_{pn}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}/\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}}{Y_{tn}}. \tag{5}$$

In the case where we have only two outputs, which could represent left and right wheel velocities in the indoor robots and the speed and the steering in the outdoor robots. The contribution made by rule p to each output will be given by S_{r1} and S_{r2}. We can calculate the contribution made by a rule to the final action given a crisp input vector as $S_c(n)=(S_{r1}+S_{r2})/2$. Given that during the episode each rule will match K crisp input vectors to different degrees then the contribution of each rule during the episode is given by:

$$S_{\mathrm{tot}}=\frac{\sum_{k=1}^{K}S_c(k)}{\sum_{m=1}^{M}S_F(m)}, \tag{6}$$

where $S_F(m)$ is the contribution of all the triggered rules which matched M crisp inputs vectors during the whole episode. Eq. (6) helps to calculate the rule contribution at the FB as they are related more to the rules that contributed through the episode and was responsible for the final failure. However at the SB we calculate the contribution of the rules that contributed from the beginning of the episode till the FB position as these rules are the rules that if their actions was correct they could helped the robot early to avoid the obstacles, their contributions is calculated as follows

$$S_{\mathrm{totSB}} = \frac{\sum_{k=1}^{K_s} S_c(k)}{\sum_{m=1}^{M_s} S_F(m)}, \tag{7}$$

K_s is the number of matched crisp input vectors from the beginning of the episode till the FB. $S_F(m)$ is the contribution of all the triggered rules which matched M_s crisp inputs vectors from the beginning of the episode till FB.

The rules that contributed most to the actions at the FB or SB positions are those rules with the greatest values of S_{tot} and S_{totSB}. In order to prevent good rules to be punished by being associated and fired by bad rules we replace first the actions of the two most effective rules at SB with the highest values of S_{totSB} as these rules will be responsible later to fire the other rules that could be good rules but could be punished by the bad actions of these effective rules till FB and cannot recover from their bad effect. We will call this the first replacement (FR). If the robot does not find a solution within a certain number of iterations, chosen empirically to be three iterations this means that modifying the SB rules is not sufficient to produce a solution and we still need to modify the most effective rules during the whole episode as their actions were bad as well. In this case the robot will modify the two most effective rules at FB with the highest values of S_{tot} and the two most effective rules at SB with the highest values of S_{totSB} we will call this second replacement (SR). As we are using delayed reinforcement at the end of the episode then the rules contribution which will be calculated according to S_{tot}.

If there was an improvement in the distance travelled by the vehicle, the rules that contributed most will be given more fitness to boost their actions. If there is no improvement then by reducing their fitness w.r.t the other rules we punish these rules and we begin examining those solutions that proposed small contributing actions.

The fitness of each rule is supplied by the *Fitness Evaluator* and is given by

$$S_{rt} = \mathrm{Constant} + (d_{\mathrm{new}} - d_{\mathrm{old}}) \,.\, S_{\mathrm{tot}} \,.\, (1-F) \,.\, V, \tag{8}$$

where d_{new} is the distance moved by the vehicle after producing a new rule-base using the online algorithm, d_{old} is the distance moved by the vehicle during the previous iteration, $d_{\mathrm{new}} - d_{\mathrm{old}}$ is the distance improvement or degradation caused by the adjusted rule-base produced by the algorithm. F is the normalised steering value. V is the normalised average speed of the vehicle. This maximises S_{rt} by increasing the distance moved by the vehicle at higher speeds with minimum steering adjustments. The variable V was introduced to favour those rules with faster speeds and F was introduced to penalise instant large steering variations and zigzag paths.

In the first population of GA, as there is no distance moved yet, we blame only those rules that have contributed most to the action of collision. The fitness of each rule is given by

$$S_{rt} = \mathrm{Constant} - S_{\mathrm{tot}}. \tag{9}$$

In this way the rules that have contributed most to this failure action will have a lower fitness value than those rules that have contributed less. Thus the GA can be directed away from bad actions that are considered ‘bad’ and encouraged to explore other ‘good’ actions.

3.1.3. Experience bank application

Zhou [49] presented a CSM (classifier system with memory) system that addresses the problem of long-term versus short-term memory, i.e. how to use past experiences to reduce the problem solving activity in novel situations. Zhou's approach is to build a system in which a short and long term memory are simultaneously present.

We propose a similar system using what we term an *Experience Bank*. Initially the vehicle has acquired no previous experience and the *Experience Bank* is empty. As it begins learning by GA, it begins filling the memory with clusters of rules. Each rule cluster consists of the rules and the actions (consequences) that were learnt using the GA. The rule clusters are stored in a queue based on recent experience, hence the term *Experience Bank*.

Each time the vehicle is presented with a situation to solve, it begins checking if the rules to be modified are present in the memory clusters or not, thus it will experiment with the memory clusters which have rules that are currently selected for replacement. The system replaces the two most effective rules actions at the FB and the two most effective rules actions at SB by rules stored in the *Experience Bank* which represents past experiences that had solved a similar situation that the robot had solved in past If, for example, we have rules 1, 2, 3, 4 to be replaced and the first cluster has the consequences of rules 1, 3, 6, 7. Then the consequences of rules 1, 3 will be changed and rules 2, 4 will remain the same. The vehicle then begins moving using the modified rule-base. If it survives and gets out of this situation without any collisions or failures, the rules are kept in the rule-base of the controller. In this way we have saved the process of learning a solution to this problem from the beginning by using the memorised experience which was developed for different environmental conditions. If the robot again fails (collides with an obstacle), it measures the distance it had moved to determine the fitness of the solution proposed by this memory cluster (supplied from the *Experience Bank*). The role of the *Experience assessor* is to determine the fitness of each memory cluster solution based on the distance it moved before collision and then proposing the solution with the highest fitness to act as a starting point for the GA search. As the experienced memory clusters are recalled in many situations and always assigned fitness according to how well they performed in each situation. The *Experience assessor* also averages the memory clusters fitness values over the number of iterations and then it ranks the memory clusters so that when a similar situation is encountered it starts with the memory cluster that had the biggest average fitness amongst the other memory clusters. This will save time as it will allow the system to experiment first with the memory clusters which performed better than the others and this can cause the robot to find a memorised solution that can solve the situation faster than going and searching through the whole Experience Bank in a nonorganised manner. However if none of the memory clusters offers a solution the robot will still experiment with all the relevant memory clusters and in case none of them offer a solution to the current problem the robots picks the best performing memory cluster (according to its fitness) to serve as a starting point for the GA search.

Starting the GA search from the best found experienced solution can be justified as follows as we often confront sets of similar problems. It makes little sense to start a problem solving search attempt from scratch with a random initial population when previous search attempts may have yielded useful information about the search space [42]. Instead, seeding a GA initial population with the best found solution to previously solved problems can provide information (a search bias) that can reduce the time taken to find a quality solution. In this case, the GA does not have to waste time exploring unpromising subspaces because these starting cases provide good building blocks for solutions to the current problem [41].

After all the memory clusters have been examined, if the robot still collides, the best solution proposed according to its fitness as determined by the *Experience Assessor* is transferred to the robot controller. This is used as the starting position for the GA search instead of starting from a random point. The Experience Bank actions, which act as a long-term memory, will thus serve to speed up the search.

A problem occurs as the system begins to accumulate experience that exceeds the physical memory limits of the *Experience Bank*. This implies that we must remove some of the stored experience information as the amount of acquired experience increases. In order to ensure that the most valuable experiences are not lost, we employ the following mechanism. To every rule cluster we attach a *difficulty counter* to count the number of iterations taken by the vehicle to find a solution to a given situation, we also attach a *frequency counter* to count how often the rule cluster has been retrieved. The *degree of importance* of each rule cluster is calculated by the *Experience Survival Valuer* and is given by the product of the *frequency counter* and the *difficulty counter*. This approach tries to keep the rules that required a lot of effort to learn (due to the difficulty of the situation) and also the rules that are used frequently.

When there is no more room in the *Experience Bank*, the rule cluster that had the least *degree of importance* is selected for replacement. If two rule clusters share the same importance degree, tie breaking is resolved by a least-recently used strategy. The rule that has not been used for the longest period of time is replaced. Thus an *age parameter* is also needed for each rule cluster. The value of the age parameter increases over time, but is reinitialised whenever the associated cluster is accessed. The limit for the memory clusters is set to 2000 rule clusters (limited by the memory capability of our vehicles). When we exceed this limit the *Experience Survival Valuer* begins using the *degree of importance* and the age operator to optimise the Experience Bank.

Our techniques are inspired from nature and Psychology theorems as it is similar to human learning where humans are born with basic information and they start learning from the best experience available in their community rather than starting to learn from scratch [36]. As in human societies, culture can be viewed as a vehicle for the storage of information that is globally accessible to all members of the society and that can be useful in guiding their problem solving activities [31]. Also humans tend to use experience to solve problems that they encounter by trying to recall their previous experience and if they need to learn a new experience they start trying to refine and tune their previous experience to solve the problem they are facing [34,36].

3.1.4. Producing new solutions by AGA

The *adaptive genetic algorithm* (AGA) is the rule discovery component for our system (as in the classifier system). The AGA is applied to learn a new solution for a certain situation, after all the solutions stored in the *Experience Bank* have failed. The AGA initialised with the best found solution from the *Experience Bank* starts to search for new consequences for the blamed rules. As was mentioned earlier, the AGA starts by modifying the actions of the most two dominant rules at the Second Backing position, we will call this the first replacement (FR). If the robot does not find a solution within a certain number of iterations, chosen empirically to be three iterations, the robot begins modifying the rules actions of the two most effective rules at FB and the two most

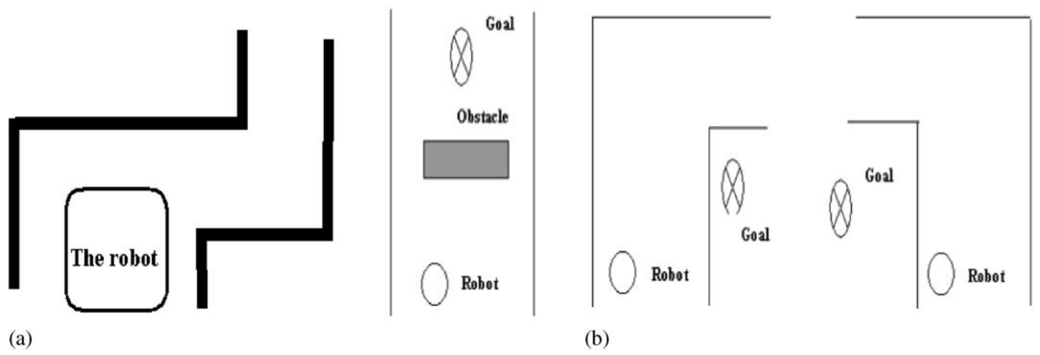


Fig. 6. (a) The robot is in a situation composed of two sub-situations, a right turn followed by a left turn. (b) Three obstacles configuration used to experiment with different mutation and crossover probabilities.

effective rules SB. This means that modifying the SB rules is not sufficient to produce a solution in this situation and the FB rules also need to be taken into consideration. We will call this Second Replacement (SR). The population of the GA during the FR will be the actions of all the rules that have contributed to the SB from the beginning of the episode till FB (which is usually a small population depending on the situation). While in the SR the population will be consisting of all the rules that contributed during the whole episode. The fitness of each rule in the first population is proportional to its contribution during the whole episode according to Eq. (9). The parents of the new solution are chosen proportional to their fitness using the roulette-wheel selection process and the genetic operations of crossover and mutation are applied to produce new solutions. After the AGA generates a new solution, the vehicle tries the controller using the modified rule-base. If the vehicle moves a distance greater than the distance needed for the ending criteria without colliding or getting very close to an obstacle, this will be considered a solution this means that the robot had learnt this particular situation. The robot keeps this solution in the rule-base and in the Experience Bank. If the robot collides again with an obstacle it goes back to the SB through the FB and calculates the fitness of each rule according to their performance during the whole episode according to Eq. (8) and then the AGA generate a new solution, which is then tried. If the number of iterations exceeds a maximum number chosen empirically to be six without a solution being found then we decrease the situation ending criteria to half the distance. This means that this situation cannot be learnt as a single situation and must be split into two smaller situations. An example is shown in Fig. 6(a). Splitting this situation into two sub-situations is essential for producing a solution.

3.1.4.1. Choice of the crossover and mutation probabilities. The crossover and mutation probabilities play a major role in the fast convergence of the GA. The crossover probability P_c controls the rate at which the solutions are subjected to crossover. The higher the value of P_c, the quicker the new solutions are introduced into the population. As P_c increases, the solutions can be disrupted faster than selection can exploit them. The choice of mutation probability P_m is critical to the GA performance. While the use of large values of P_m transforms the GA into a purely random search algorithm, some mutation is required to prevent the premature convergence of the GA to sub optimal solutions.

The reduced crossover lowers the productivity of the GA, since there is less recombination between individuals and hence it takes a longer time to obtain a good solution, so as in [25] we set the crossover probability to 1.0 to guarantee fast convergence.

The traditional role of mutation has been that of restoring lost or unexplored genetic material into the population to prevent the premature convergence of the GA to sub optimal solutions. However recent investigations have demonstrated that high levels of mutation could form an effective search strategy when combined with conservative selection methods [40]. As we are using a small population size and small chromosome size, it is necessary to use a high mutation rate to allow for a wider variation in the search and hence the ability to jump out of any local minima without reducing the search process to a random process [39]. It is also desirable that as the system fitness improves (distance traveled increases), the mutation rate is decreased so as not to lose the genetic material that caused the improvement.

Therefore, we vary the mutation probability from one generation to another depending on the distance improvement achieved. In the first few iterations we want to use a high mutation probability. If there is an improvement in the distance, we will linearly reduce the mutation until the improvement reaches a maximum value and the mutation probability reaches 0. If the distance improvement was the same or was less than the previous trials, then the mutation probability is increased again to a high value to find new genetic materials that might aid in finding a solution. Thus, the mutation probability can be written as

$$\begin{aligned} &p_{\mathrm{m}} = H - \frac{H(d_{\mathrm{new}} - d_{\mathrm{old}})}{\text{robot length}} \quad \text{if } \boldsymbol{d}_{\mathrm{new}} > \boldsymbol{d}_{\mathrm{old}}, \\ &P_{\mathrm{m}} = 0 \quad \text{if } (d_{\mathrm{new}} - d_{\mathrm{old}}) \geqslant \text{robot length}, \\ &p_{\mathrm{m}} = H \quad \text{otherwise.} \end{aligned} \tag{10}$$

So what we want is to find the best value for H that will help to speed the convergence without causing the search to end by a random search or being trapped in a local minima. As all the algorithm parameters are related to the vehicle dimensions thus we have related the maximum distance improvement to the robot length. This choice will be verified through practical experiments.

In order to find the best value for H we had performed practical experiments with real robots having different sizes and kinematics using the experimental set-up in Fig. 6(b). As the obstacle avoidance behaviour cannot be employed as an independent behaviour (asking the robot to wander randomly and not to collide). The obstacle avoidance behaviour can be regarded as a safety behaviour associated with other behaviours. In the following experiments, we will coordinate the obstacle avoidance behaviour with the goal seeking behaviour whose rule-base was obtained from our previous work [12,13,16]. In the first set of experiments we wanted to verify our choice for the crossover probability and find the best H value for the adaptive mutation probability in Eq. (10). For each experiment, we experimented with a value for the crossover values and a value for H for the adaptive mutation in Eq. (10).

We tried six different values of crossover probability starting from 0 to 1 with a step of 0.2, each with a value for H varying from 0 to 1 with values equal to 0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0. We kept the robot length parameter fixed in these sets of experiments however, we will verify later its choice. For each crossover probability value and H value (which is related to the mutation value)

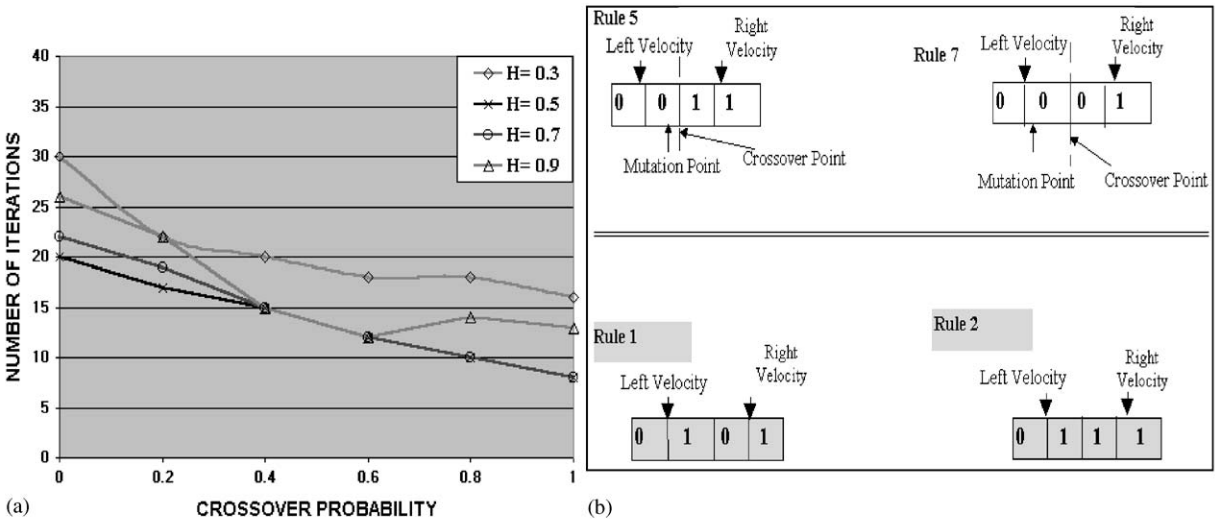


Fig. 7. (a) The convergence rate specified by the number of iterations plotted against the crossover probability for different H values. (b) The rule genotype and an example of GA processes in our system in which rule 5 and rule 7 (which were selected due to their higher fitness values) are generating new consequent for rules 1,2 of the SB reversal. The same will happen with two FB rules.

we tried many experiments with the three experimental configurations shown in Fig. 6(b). We had performed 5 experiments for each configuration shown in Fig. 6(b) starting from different random points and changing the sizes of the obstacles and the dimensions of the configuration and using different sizes of robots with different kinematics. We recorded the number of iterations taken by the robot to find a solution for each experiment and we calculated the average number of iterations over the five experiments. We compared the results for the three configurations and we found that they all gave the same patterns shown in Fig. 7(a).

It was noted that at zero H value and thus zero mutation probability no solution could be found because of the lack of genetic material, the same was true for a value of H value of 0.1. At an H of 0.3, the fastest convergence was after 16 iterations with crossover value of 1.0. At an H value of 0.5, we have the fastest convergence after 8 iterations with a crossover rate of 1.0. The same convergence rate was noted for H values of 0.7. At H value of 0.9 the system more or less performs a random search and the best performance is at crossover probability of 1.0 after 6 iterations. The H value of 1.0 leads to no solutions at all. This is because starting with most of the genetic material the same, any mutation will lead to an inversion of the binary material and thus will degenerate to a random search as no new material will be introduced.

From Fig. 7(a) it is obvious that as the crossover probability increases, the convergence rate also increases with an optimum value achieved at a crossover probability of 1.0. It also shows that optimum H value to be either 0.5 or 0.7, but we will choose 0.5 to be the upper bound to decrease the risk of ending in a randomised search.

We conducted another series of experiments to investigate the optimum parameters in the base of Eq. (10). We used the same three experimental configurations shown in Fig. 6(b). We had performed five experiments for each configuration shown in Fig. 6(b) starting from different random points and changing the sizes of the obstacles and the dimensions of the configuration using different sizes of robots with different kinematics. We recorded the number of iterations taken by the robot to find a solution for each experiment and we calculated the average number of iterations over the five experiments. We tried various lengths relative to the robot length such as half or double its original length. These experiments will determine if the *robot length* in the base of Eq. (10) was the best value?. We used the best values found from Fig. 7(a) where $H=0.5$ and the crossover probability $=1.0$. It was found that the original robot size had produced the fastest convergence whilst any other length did not produce the same fast convergence. So by substituting with best found H values Eq. (10) can be written as

$$p_{\mathrm{m}} = 0.5 - \frac{0.5(d_{\mathrm{new}} - d_{\mathrm{old}})}{\text{robot length}} \quad \text{if } \boldsymbol{d}_{\mathrm{new}} > \boldsymbol{d}_{\mathrm{old}},$$

$$P_{\mathrm{m}} = 0 \quad \text{if } (d_{\mathrm{new}} - d_{\mathrm{old}}) \geqslant \text{robot length},$$

$$p_{\mathrm{m}} = 0.5 \quad \text{otherwise.}$$

We use binary coding in the AGA chromosomes. If we again use the case where there are two actions for each rule in this case the actions represent left and right wheel velocities in case of indoor vehicles and speed and steering in case of outdoor vehicles. Assuming we have 4 output membership function, we can decode each action by two bits as follows, *Very Low* as 00, *Low* as 01, *Medium* as 10, *High* as 11. By doing this we have a chromosome length of 4 bits.

Fig. 7(b) shows a description of the AGA operations with the rules genotype in which rule number 5 of the obstacle avoidance and rule number 7 are chosen for reproduction by roulette wheel selection due their high fitness. They have been identified as having contributed most to the final action. The crossover probability of 1.0 was applied to both chromosomes and the adaptive mutation as well. The resultant offsprings were used to replace the consequents of rules 1 and rule 2 which were mostly blamed at SB. The same technique can be used to replace the consequents of the two dominant rules in the FB.

3.1.4.2. Contextual constraints. The incorporation of evolutionary computation (EC) and knowledge-based mechanisms can considerably improve the EC performance when common knowledge is used to bias the problem solving process [30,31,43]. The knowledge, such as using some constraints can be used to guide the generation of candidate solutions, promote more instances of desirable candidates, and reduce the number of less desirable candidates in the population. Using constraints is a known technique to narrow the search space and speed up the search for GA [31,43]. Using the *Contextual Constraints* will not reduce the significance of the learning approach as the system can still learn without *Contextual Constraints* but it will take longer time, however in on-line learning in unstructured environments (e.g. outdoor environments) we want to use all means to speed up the search to a good solution. In order to speed up the search we will use the *Contextual prompter Constraints*. This uses sensor information in order to narrow the GA search space to make it avoid regions which will not provide any solutions. For example, it is not a good idea to turn left when the sensors detect that the left end is blocked or that there is larger space to turn right. It was found through practical experiments that when the algorithm is not using *Contextual Constraints* the system needs on average five times the amount of time required when using *Contextual Constraints*.

This is due to the system searching through the whole search space and not only in the regions where solutions are likely to be found.

The *Contextual Constraints* also contains basic knowledge that prevents generating dangerous rules that can damage the robots or the environment thus also ensuring the safety of the generated behaviour. For example the *Contextual Constraints* will avoid generation of rules that will allow the robot to go with maximum speed when there is an obstacle in its way and very close to it.

3.1.5. Ending criteria

At the SB the vehicle is at a minimum distance of $2W$ from the front obstacle and at a minimum distance of X from the left and right obstacles. If it is assumed that the vehicle moves a distance W without taking any correct action and at the position of FB it makes a maximum turn. This will cause the vehicle to rotate in some how a quarter of a circumference of the circle with a radius W making the robot move $\pi W/2$. In order to make sure it is out of this situation without colliding, the vehicle should further travel a distance of X. Therefore the total distance moved by the vehicle to get out safe of a situation without hitting an obstacle can be given by

$$W + \frac{\pi W}{2} + X. \tag{11}$$

So the vehicle can calculate the distance moved, if this distance exceeds the distance given by Eq. (11), then the vehicle has successfully found a solution to this situation. If the number of generations exceeds 6 without successfully producing a solution, the distance given by Eq. (11) is reduced by half. This sub-divides the problem allowing each situation to be solved separately.

3.2. Online adaptation of the learnt controller and life long learning

In the previous sections we described how, using the obstacle avoidance behaviour as an example, behaviour rule bases that receive delayed reinforcement can be learnt online. In this section we discuss how the vehicle can adapt to environmental changes and how a life long learning scheme of the coordinated behaviours can be implemented, which allow the robot to add to or update its knowledge when encountering new situations. Thus the coordinated rule bases used by the vehicle becomes dynamic, allowing rules to be added, deleted or modified according to the situations that are encountered. This approach is useful in prototyping as we can learn the robotic controller on a prototype robot in a controlled environment and then we transfer it to the big robot running in the real environment where we only run a short adaptation cycle [16]. As the algorithm parameters are vehicle independent, the algorithm can easily be moved between the prototype and the real vehicle requiring only minimal changes. This procedure is useful in learning controllers for vehicles which might involve dangerous manoeuvres using small and cheap prototype robots in a controlled environment, thus avoiding the risks associated with online learning using heavy and expensive vehicles. Also these techniques produce general controllers that can be applied to different types of vehicle having the same sensor configurations and performing the same mission after applying a short adaptation cycle. Thus saving the need to generate a new controller for each different vehicle. Perhaps more importantly, this adaptation session is important in the case where environmental or vehicle kinematics changes occur and the vehicle needs to rapidly adapt to these changed circumstances. We will validate these techniques through our experiments in challenging environments such as the agricultural environments.

Although fuzzy logic allows a degree of imprecision to exist within the environment, significant environmental or robot differences will prevent the rule sets from operating correctly. One solution to this problem is to provide a new rule set for each of the different environments, although prior knowledge of the different environments would be needed (which could be difficult, if not impossible, for some environments). An alternative solution is to allow the existing coordinated rule set to be adapted to compensate for the environmental differences. This latter approach only requires a basic rule set to be present in the prototype controller and does not require any prior knowledge of the possible environments.

We start the adaptation using the HFLC that best fits the current situation, instead of starting from a random point. If the system were started using a random rule base, the modification of the actions would take approximately six times longer than starting from a known 'good' rule base containing some inappropriate rules for the new environment. This figure was found by practical experimentation.

In our case the adaptation is performed by modifying rules in (in different behaviours) that, if modified, would lead to the vehicle adapting to its new environment. Whilst it might seem a good idea to modify the coordination parameters or the membership functions of the individual behaviours to adjust the robot behaviour when it fails, this is not the best approach. This can readily be illustrated by considering a case where the rules in the individual rule bases become inappropriate, it is obvious that changing the coordination parameters in these circumstances would never correct the vehicle behaviour (as was proved by experimentation). Also changing the MF is a difficult task, as this needs to be done for each individual behaviour, necessitating that the robot be taken away from its environment for MF calibration. However, our method allows the poorly performing rules for each behaviour to be found and corrected, online, modifying only the actions of a small number of rules that performed poorly.

3.2.1. Summary of the steps and parameters used in online adaptation of the coordinated behaviours

In this section we give an overview of the steps and parameters used for online adaptation of the co-ordinated behviours.

(1) The robot mission is composed of many tasks, each task has a measure of success attached to it. If one or more of tasks in the robot mission fails down, the adaptation cycle starts. The failures of tasks like obstacle avoidance is triggered when it collides or gets very close to an obstacle, while the performance of behaviours that receive immediate reinforcement are evaluated over a distance equivalent to twice the robot length and if their performance falls below their specified target, then the failures of these task (s) is triggered. The system does not need to relearn the whole rule bases to adapt to this failure, it will adjust rules in the different coordinated behaviours that if adjusted will cause the robot to recover from the failure situation The system can adapt any time when it fails. This leads to a life long learning approach, where the coordinated rule bases used by the vehicle becomes dynamic, allowing rules to be added, deleted or modified according to the situations that are encountered.

(2) The robot uses similar STM as explained in Section 3.1.1 to return to its pre failure position to find the rules in the different coordinated behaviours responsible for failure and correct them to generate a solution to the current failure. If one of the failing tasks is obstacle avoidance the system backs first a distance which is maximum of the distance equivalent to the robot length and the distance sufficient to satisfy the minimum safe distance W from the front and X from the sides of the vehicle as explained in Section 3.1.1. If none of the failing tasks is the obstacle avoidance it backs a distance equivalent to the robot length. This is called the FB and then backs to double the FB distance called the SB. These two distances allows to discovery of the blamed rules for failure which if corrected could solve the situation.
(3) The system finds the two most effective rules at SB with the biggest values of $S_{\text{tot}c\text{SB}}$ using Eq. (15) and the two most effective rules at FB with the biggest values of S_{tot} using Eq. (14) as explained in Section 3.2.2.
(4) The system then replaces the two most effective rules (which can belong to different behaviours) at the FB and SB by rules stored in the *Experience Bank* which represents past experiences that had solved a similar situation encountered in past.
(5) The *Experience Bank* solutions are tested one by one in a pre specified order starting from the most successful rule clusters over the past experiences. If one of the experiences solves the situation then the learning ends and the rule base continues with this memorised experience with no need to generate new solutions.
(6) If all the solutions from the *Experience Bank* fail, the system allocates each experienced solution a fitness value according to how well it performed in the environment. The AGA starts its search from the experienced solution with the best fitness as evaluated by *Experience Assessor*. Thus we do not start our learning from random but from the best point in search space, this action helps to speed up the search.
(7) In order to prevent good rules to be punished by being associated and fired by bad rules we replace first the actions of the two most effective rules at SB (which can belong to different behaviour) with the highest values of $S_{\text{tot}c\text{SB}}$ as these rules will be responsible later to fire the other rules that could be good rules but could be punished by the bad actions of these effective rules till FB and cannot recover from their bad effect. We will call this the First Replacement (FR). The AGA population will consist of all the active rules from the beginning of the episode till the FB. We will use AGA to generate new solutions, the rule fitness will be calculated according to Eq. (16). The system will aim to recover the failed tasks while still achieving the other tasks objectives (that were active during the episode) according to their level of activation during the episode. The system will also use the *Contextual Constraints* explained in Section 3.1.4.2 to narrow the search space.
(8) The vehicle will move to test the solution. If it solves the situation by satisfying the ending criteria by recovering the failed tasks and achieving the goals of the other activated tasks during the episode. This is assumed as a solution and the learning cycle ends and it stores this solution in the *Experience Bank*. If the robots fails and the number of iterations is less than 3 the vehicle returns to SB through FB and returns to step 7. If the number of iterations is bigger than 3 and the robot fails goto step 9.
(9) If the robot does not find a solution within a certain number of iterations, chosen empirically to be three iterations, the robot begins modifying the rules actions of the two most effective rules at FB and the two most effective rules SB. This means that modifying the SB rules is

not sufficient to produce a solution in this situation and the FB rules also need to be taken into consideration. We will call this SR. The population will consist of all the active rules during the episode. The rule fitness will be calculated according to Eq. (16). The AGA will be used to generate new solutions and we will use *Contextual Constraints* explained in Section 3.1.4.2 to narrow the search space.

(10) The vehicle then starts moving with the modified rules. If the vehicle satisfies the ending criteria this will be considered a solution and the learning cycle ends and it stores this solution in the *Experience Bank*. If it fails the vehicle returns to SB through FB and returns to step 9.

Fig. 8 show a flowchart of the steps used for online adaptation and life long learning.

3.2.2. *The techniques in details*

In robot navigation the designer will define a high level mission which will be a mixture of different tasks such as *achieve goals while avoiding obstacles*, each task will have a measure of success attached to it. If any of the tasks fails down, their failure will trigger the adaptation cycle to recover from this failure. The failures of tasks like obstacle avoidance is triggered when it collides or gets very close to an obstacle, while the performance of behaviours that receive immediate reinforcement are evaluated over a distance equivalent to twice the robot length and if their performance falls below their specified target, then the failures of these task (s) is triggered. The robot Performance (P) is a weighted function, in which robot is trying to achieve the various tasks according to their importance. The failed tasks performances will be the most important for the robot to satisfy and recover from failure and thus they will be weighted by one, while the other tasks performances will be weighed by how much they were activated during the whole episode. The robot overall fitness function is maximised by improving the robot performance (P).

When one or more tasks of the robot mission fail down the adaptation cycle is activated. The robot uses similar STM as explained in Section 3.1.1 to return to its pre failure position to find the rules in the different coordinated behaviours responsible for failure and correct them to generate a solution to the current failure. The robot first performs the FB as follows: If one of the failing tasks is obstacle avoidance the system backs first a distance which is maximum of the distance equivalent to the robot length and the distance sufficient to satisfy the minimum safe distance W from the front and X from the sides of the vehicle as explained in Section 3.1.1. If none of the failing tasks is the obstacle avoidance it backs a distance equivalent to the robot length. The robot then backs to double the FB distance called the SB.

At the FB the robot finds the two most dominant rules (which can belong to two different behaviours) that contributed mostly through the episode and were responsible for the final failure. It then returns to the SB position and finds the two most dominant rules (which can belong to different behaviours) from the beginning of the episode till the FB position as these rules are the rules that if their actions was correct they could helped the robot early to avoid failure. The contribution of each rule to the final output in a given situation as follows: Apply Eq. (2) and substitute Y_t from Eq. (1). Then we can write the crisp output Y_{ht} as

$$Y_{ht} = \frac{\sum_{y=1}^{B} mm_y(\sum_{p=1}^{M} y_p \prod_{i=1}^{G} \alpha_{A\mathrm{ip}} / \sum_{p=1}^{M} \prod_{i=1}^{G} \alpha_{A\mathrm{ip}})}{\sum_{y=1}^{B} mm_y}, \tag{12}$$

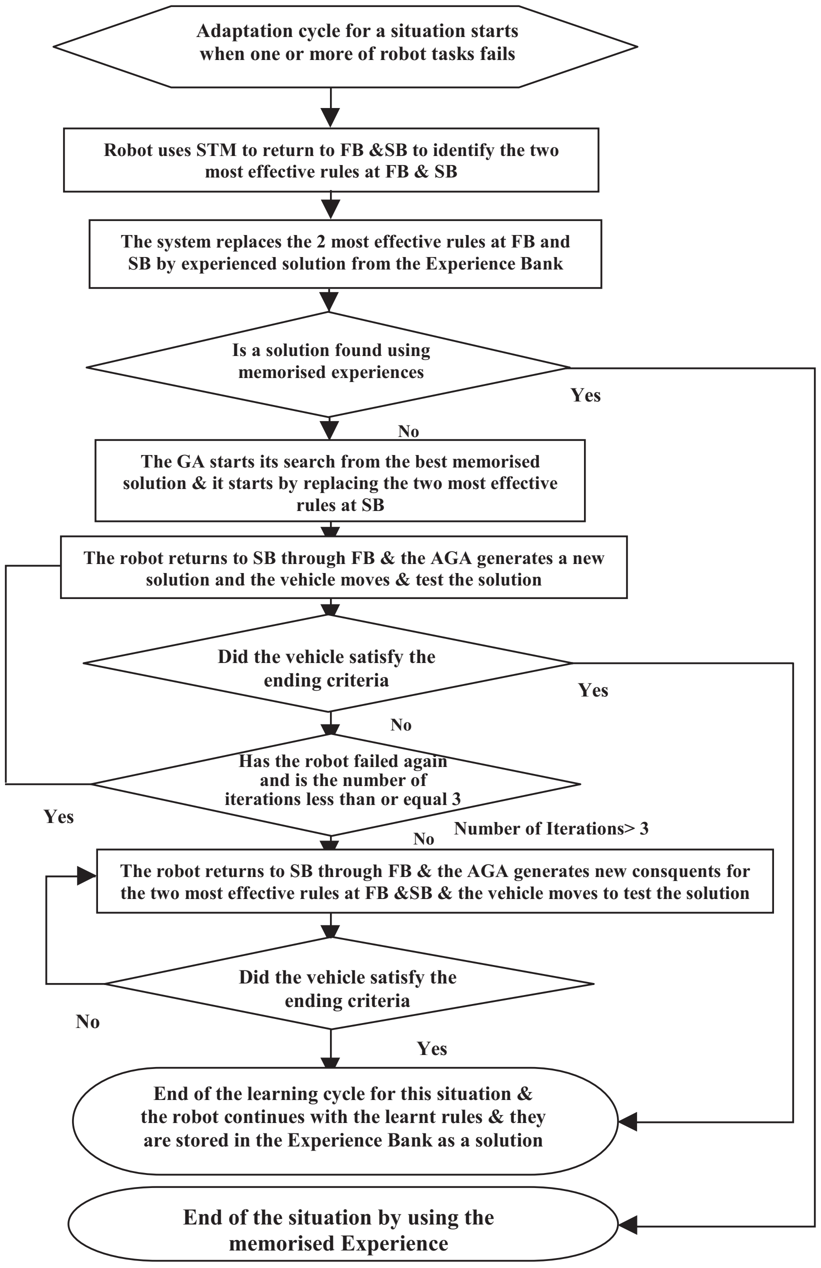

Adaptation cycle for a situation starts when one or more of robot tasks fails
Robot uses STM to return to FB &SB to identify the two most effective rules at FB & SB
The system replaces the 2 most effective rules at FB and SB by experienced solution from the Experience Bank
Is a solution found using memorised experiences
Yes
No
The GA starts its search from the best memorised solution & it starts by replacing the two most effective rules at SB
The robot returns to SB through FB & the AGA generates a new solution and the vehicle moves & test the solution
Did the vehicle satisfy the ending criteria
Yes
No
Has the robot failed again and is the number of iterations less than or equal 3
Yes
No
Number of Iterations> 3
The robot returns to SB through FB & the AGA generates new consquents for the two most effective rules at FB &SB & the vehicle moves to test the solution
Did the vehicle satisfy the ending criteria
No
Yes
End of the learning cycle for this situation & the robot continues with the learnt rules & they are stored in the Experience Bank as a solution
End of the situation by using the memorised Experience

Fig. 8. Flow chart of steps used in online adaptation and the life long learning strategy.

where M is the total number of rules, Y_p is the crisp output for each rule, $\prod \alpha_{A\mathrm{ip}}$ is the product of the membership functions of each rule inputs. G is the number of the input variables, mm_y is firing strength of each of the four behaviours. B is the number of the activated behaviours.

If we have N output variables, then we have $Y_{ht1}, Y_{ht2}, \dots, Y_{htn}$. The contribution of each rule p in the behaviour y to the total output $Y_{ht1}, Y_{ht2}, \dots, Y_{htn}$ is denoted by $S_{ra11}, S_{ra22}, \dots, S_{rann}$, where $S_{ra11}, S_{ra22}, \dots, S_{rann}$ are given by

$$S_{ra11} = \frac{(mm_y/\sum_{y=1}^{B} mm_y).(Y_{p1}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}/\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}})}{Y_{ht1}},$$

$$S_{ra22} = \frac{(mm_y/\sum_{y=1}^{B} mm_y).(Y_{p2}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}/\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}})}{Y_{ht2}},$$

$$S_{rann} = \frac{(mm_y/\sum_{y=1}^{B} mm_y).(Y_{pn}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}}/\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{A\mathrm{ip}})}{Y_{htn}}. \tag{13}$$

We then calculate each rule's contribution to the final action Sa_{c1} by

$$Sa_{c1} = \frac{S_{ra11} + S_{ra22} \cdots + S_{rann}}{N}.$$

In the case where there are only two output variables, we have $Sa_{c1} = (S_{ra11} + S_{ra22})/2$. Given that during the episode each rule in each coordinated behaviour will match K crisp input vectors to different degrees then the contribution of each rule during the episode is given by

$$S_{\mathrm{tot}c} = \frac{\sum_{k=1}^{K} Sa_{c1}(k)}{\sum_{m=1}^{M} Sa_F(m)}, \tag{14}$$

where $Sa_F(m)$ is the contribution of all the triggered rules in the different coordinated behaviour which matched M crisp inputs vectors during the whole episode. Eq. (14) helps to calculate the rule contribution at the FB as they are related more to the rules that contributed mostly through the episode and was responsible for the final failure. However at the SB we calculate the contribution of the rules that contributed mostly from the beginning of the episode till the FB position as these rules are the rules that if their actions was correct they could helped the robot early to avoid the obstacles, their contributions is calculated as follows:

$$S_{\mathrm{tot}c\mathrm{SB}} = \frac{\sum_{k=1}^{K_s} Sa_{c1}(k)}{\sum_{m=1}^{M_s} Sa_F(m)}, \tag{15}$$

K_s is the number of matched crisp input vectors from the beginning of the episode till the FB. $Sa_F(m)$ is the contribution of all the triggered rules in the different coordinated behaviours which matched M_s crisp inputs vectors from the beginning of the episode till FB.

The rules that contributed most to the actions at the FB or SB positions are those rules with the greatest values of $S_{\mathrm{tot}c}$ and $S_{\mathrm{tot}c\mathrm{SB}}$.

In order to implement life-long learning the vehicle utilises its *Experience Bank explained before*, which is composed of all the experiences encountered previously and the solutions generated. The *Experience Bank* is the same as explained above apart from that each cluster will contain the rules and which behaviours they belong to and their learnt actions. These clusters represents how the agent has adapted to the different changing environments and situations, i.e. in two different rule clusters we can find the same rule having different consequents which means that each environment should have different actions.

The robot matches the two most effective rules during the FB and the two most effective rules during the SB against the sets of rules stored in the *Experience Bank*. If, for example; rules 1, 2 from the obstacle-avoidance rule base, rule 3 of the left edge-following behaviour and rule 4 of the right edge following need to be replaced (most effective rules in FB and SB), AND the first rule cluster in the Bank contains the consequences of rules 1 of obstacle avoidance; rule 3 of left edge following and rules 6, 7 of right edge following, then the consequences of rules 1 for obstacle avoidance and 3 for left edge following will be changed and rule 2 for obstacle avoidance and 4 for right edge following will remain the same. Then the robot starts operating with this modified rule-base taken from the *Experience Bank*. If it survives and gets out of this situation with no failures, then these rules are kept in the rule-base of the controller. In this way we have saved the process of learning a new solution to this problem by using an experience that had previously worked in different environmental conditions. This is the same as in nature, where the agent tries to recall its previous experiences in an attempt to solve similar situations [36]. If none of the experiences had solved this situation, the vehicle assigns a fitness value to each experience indicating how well it performed using the *Experience Assessor* as explained in Section 3.1.2. The vehicle then chooses the consequents of the rule clusters that have given the highest fitness and keeps them in the controller rule-base. These then serve as a starting position for the adaptive genetic algorithms (AGA) search instead of starting from a random point. As explained above in Section 3 this action helps to speed up the genetic search as it starts the search from the closest point in the search space.

As the robot mission is composed of various tasks and each task will have a measure of success attached to it. The robot performance function (P) is a weighted function, in which robot is trying to achieve the various tasks according to their importance in influencing and improving the robot performance. The tasks that failed and triggered the adaptation cycle will be the focus of the robot to adapt and recover them thus their performance will be weighted by one as it is very important for the robot to improve its performance for the failing task. The other tasks performances will be weighed by how much they were activated during the whole episode. For example suppose a robot had the high level mission of following an irregular edge and achieving a goal at the end of this edge while avoiding obstacles, if the robot collided with obstacles then the overall robot performance function (P) will be composed of the normalised distance moved before collision and the weight for this is one. Depending how the edge following behaviour and the goal seeking were activated during the whole episode their performances will be taken into consideration according to their average level of activation during the episode so the collective performance function (P) can be written as P is the normalised distance moved before failure, ^{*1} the normalised deviation from a goal, $^*mm_{\text{goal}}$—normalised deviation from the desired edge following distance $^*mm_{\text{following}}$, where $mm_{\text{following}}$ is the average activation level of the edge following behaviour during the whole episode and mm_{goal} is the average activation level of the goal seeking behaviour during the whole episode. This makes the Performance maximised by maximising the normalised distance the robot moves before failure to improve the obstacle avoidance behaviour. Also the performance is maximised by decreasing the Normalised deviation from the goal and from the edge to improve the goal seeking and the edge following behaviours respectively. The $+$ and $-$ signs were used to symbolise how the performance is maximised by maximising some parameters (when using the $+$ sign) and minimising other parameters (when using the $-$ sign). In case what triggered the adaptation cycle was that the robot had failed to follow the edge for a specified distance within a pre specified range then the robot overall performance function (P) will be composed of the deviation from the edge and this will be multiplied by a weight of one and the distance from obstacles and this will be multiplied by the activation level of the obstacle avoidance behaviour and the deviation from the goal multiplied by the activation level of the goal seeking behaviour during the whole episode.

The vehicle calculates the fitness value of the solution as follows if there is an improvement in the Performance function (P) produced by the modified rule base, then the rules that contributed most must be given increased fitness to boost their actions. If there is no improvement then the rules that contributed most are punished by reducing their fitness and the solutions that had small contributing actions are examined. The fitness of each rule is supplied by the *Solution Evaluator* and is given by

$$S_{\mathrm{rat}\ 1} = \mathrm{Constant} + (P_{\mathrm{new}} - P_{\mathrm{old}}).S_{\mathrm{tot}c}.(1 - F).V, \tag{16}$$

where P_{new} is the new performance function of the vehicle using the modified rule base, P_{old} is the old performance function of the vehicle before modifying the rule base, $P_{\mathrm{new}} - P_{\mathrm{old}}$ is the performance improvement or degradation caused by the adjusted rule-base produced by the algorithm. F is the normalised vehicle steering. V is the normalised average speed of the vehicle. This makes S_{rt} maximised by improving the vehicle performance at higher speeds and with minimum adjustments to the steering. The variable V was introduced to favour those rules that produced faster speeds and F was introduced to penalise instant large steering variations and zigzag paths. In the first population of the AGA, because there is no performance assessed yet, we blame the rules that have contributed most to the behaviour failure. The fitness of each rule is given by

$$S_{rat} = \mathrm{Constant} - S_{\mathrm{tot}c}. \tag{17}$$

The rules that contributed most to the actions at the FB or SB positions are those rules with the greatest values of $S_{\mathrm{tot}c}$ and $S_{\mathrm{tot}c\mathrm{FB}}$. In order to prevent good rules to be punished by being associated and fired by bad rules we first replace using the AGA the actions of the two most effective rules at SB with the highest values of $S_{\mathrm{tot}c\mathrm{SB}}$ as these rules will be responsible later to fire the other rules that could be good rules but could be punished by the bad actions of these effective rules till FB and cannot recover from their bad effect. We will call this the FR. If the robot does not find a solution within a certain number of iterations, chosen empirically to be three iterations this means that modifying the SB rules is not sufficient to produce a solution and we still need to modify the most effective rules during the whole episode as their actions were bad as well. In this case the robot will modify the two most effective rules at FB with the highest values of $S_{\mathrm{tot}c}$ and the two most effective rules at SB with the highest values of $S_{\mathrm{tot}c\mathrm{SB}}$ we will call this SR. As we are using delayed reinforcement at the end of the episode then the rules contribution which will be represented by $S_{\mathrm{tot}c}$.

The parents of the new solution are chosen proportional to their fitness using the roulette-wheel selection process and the genetic operations of crossover and adaptive mutation are applied. The fitness of each rule in the first population is proportional to its contribution during the whole episode according to Eq. (17). Binary coding is used in coding of the chromosomes. The population of the GA during the FR will be the actions of all the rules that have contributed to the SB and were active from the beginning of the episode till FB (which is usually a small population depending on the situation). While in the SR the population will be consisting of all the rules that contributed during the whole episode.

If for example rule number 5 of the obstacle avoidance and rule 7 of the left wall following are chosen for reproduction by roulette wheel selection due their high fitness. This implies they have either contributed more with their actions to the final collective action that caused improvement, or contributed less with their actions to final action that caused degradation. Then we apply adaptive crossover and mutation operators to both chromosomes. The resultant offspring will be used to replace the consequent of rules 1 and 2 for obstacle avoidance that were largely blamed for the robot failure. The AGA is constrained to produce offspring according to the *Contextual Constraints* according to Section 3.1.4.

After the AGA generate new consequents using the AGA for the blamed rules, it tests the new solution for a certain time interval in which the robot performance is measured in order to recover from the failure situation. *The robot succeeds in solving the situation and recovers from failure when it recovers the failing task*(*s*) *and satisfies the other activated task objectives.* In this case the robot keeps this solution in the rule-base and in the Experience Bank. If the robot fails again it goes back to the SB through the FB and calculates the fitness of each rule according to their performance during the whole episode according to Eq. (16) and then the AGA generate a new solution and it tries it. However, a problem occurs as the system begins accumulating experience that exceeds the physical memory limits. We are going to use the same technique used for the *Experience Survival Evaluator* mentioned in Section 3.1.3.

4. Experiments and results

We have used different vehicles to conduct our experiments, a small indoor robot steered via its driving wheels and two large outdoor robots with a conventional steering system. The indoor and the outdoor robots will have the same sensors. Each robot is equipped with three forward-looking sensors for obstacle avoidance which are the left front sensor (LFS), the middle front sensor (MFS) and the right front sensor (RFS). There are two sonar sensors on each side of the robot for edge following which are right side front (RSF) sensor and right side back (RSB) sensor for the right edge following behaviour and left side front (LSF) sensor and left side back (LSB) sensor for the left edge following behaviour. In most of the experiments the sonar sensors will be represented by three membership (which are near, medium, far) as this was found to be the least number of membership function to give a satisfactory result. Also the robots have sensors to measure the bearing from the goal, which will be infrared scanners in the indoor robots and GPS and electronic compass in the outdoor robots. Each bearing sensor is represented by seven memberships (which are very very negative, very negative, medium negative, zero, medium positive, very positive, very very positive) as this was found to be the least number of membership functions to give a satisfactory result. The output membership functions in most experiments will be represented by four membership functions (which are very low, low, medium and high). For the outdoor vehicles, in case of the steering the very low and low will turn left while the medium will be go straight ahead and the high will be turning right.

In other work based on simulation [24,26], the problem of the robot returning to the same position is completely ignored because the physical process is by-passed. While in training a real robot, the majority of the time is consumed by moving along the geometrical structure and applying the STM and triangulation and applying the pilot fuzzy controller to bring the robot to approximately to the same starting situation each trial and testing new solutions, while generating new solutions takes less that 5% of the whole learning time. Thus, when comparing our work with that of other researchers, we compare the number iterations needed to find a solution.

In experimenting with real robots it is very hard to estimate the deviation from desired values using real robots especially in outdoor environment which involves dealing with irregular edges whose borders cannot be defined with big precision. However we use the data returned from the sonar sensors and then cross validate the data by comparing the acquired data with robot response. The indoor robot response was drawn using a paint bottle fixed to the right back corner of the robot while for the outdoor robots the robot response was drawn using a tape fixed under the left back wheel. If for example we wanted to measure the robot deviation from a certain edge (wither the robot wanted to avoid this edge or follow it) we can record the average deviation reading from the edge as measured by the sonar over the whole trial and then we cross validate these values with the values we get from drawing the robot path related to the edge. Although this doesn't give us precise values but it gives an idea about how the robot had performed its mission.

For experiments learning the obstacle avoidance behaviour the preliminary input membership function (MF) are supplied by the designer. The optimum value of these MFs was learnt in [15]. The robot can learn the obstacle avoidance rule-base by encountering different situations (local solutions) while it navigates as was explained in Section 3. Then the interaction among local models, due to the intersection of neighbouring fuzzy sets causes the local learning to reflect on the global performance. Thus the robot does not need to learn special situations, but rather it learns general rules, such as *if the right sensor is low and the medium sensor is low and the left sensor is high then go left*. By encountering many different situations the robot can fill its rule-base. So the training environment should be as complex as possible to supply the robot with as many different learning situations as possible. In addition the robot only learns when it is necessary. For example, if the robot was launched into a corridor, it will learn the rules needed to navigate in this corridor, these rules can then be generalised, as will be shown later, to allow the robot to navigate in different shapes of corridors. However, when the robot is introduced to a more complicated situation, such as a maze, it must learn additional rules in order to survive. This also means that if after learning a complete rule-base the robot kinematics or the ground conditions change, the robot can still adapt itself to the environment by adjusting only a small subset of the rules.

The problem with using the obstacle avoidance behaviour is that it cannot be employed as an independent behaviour (asking the robot to wander randomly and not to collide). The obstacle avoidance behaviour can be regarded as a safety behaviour associated with other behaviours. For example, the robot can be asked to reach a target while avoiding obstacles or following an edge while avoiding obstacles. In the learning experiments that follow we will co-ordinate the obstacle avoidance behaviour with the goal seeking behaviour whose rule-base was obtained from our previous work [12,13,16]. Throughout the experiments dealing with learning the rule base for the obstacle avoidance behaviour the co-ordination context rule will be reduced to: *IF the Minimum Front Distance* ($d1$) *IS LOW THEN OBSTACLE AVOIDANCE*, *IF the surrounding is obstacle free* ($d4$) *IS HIGH THEN GOAL SEEKING*. The obstacle avoidance rule-base is initialised randomly (but safely) to move the robot forward (in any direction), this action is important in order to ensure that the robot is moving and does not remain stationary.

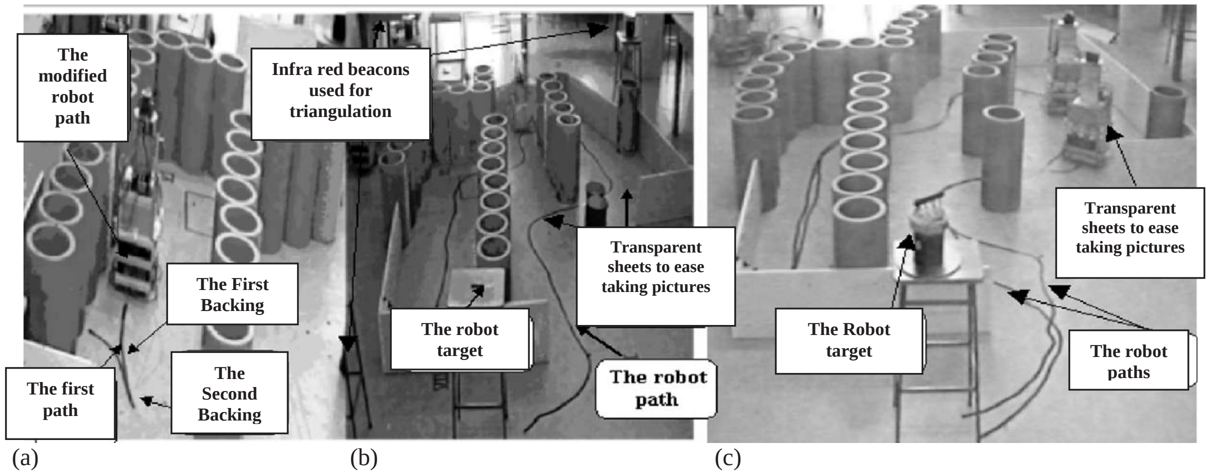


Fig. 9. (a) The robot learning cycle. (b) The robot path after learning. (c) Two different robots paths with two learnt rule-bases.

The robot learning cycle and the experimental set-up with the triangulation beacons are shown in Fig. 9(a). In this example the robot moves and fails, it then returns to the FB and then SB. After trying the *Experience Bank* solutions which fail, it then tries other solutions generated by the AGA and finally it generates a modified set of rules to solve the situation. It can then pass safely until other rules fail again, when the learning cycle will be repeated. The robot was then introduced to the complicated geometrical structure shown in Fig. 9(b). This contains many different general situations, such as how to navigate in a corridor, how to do a left turn, how to do a right turn, and how to navigate in wide areas with dead ends.

After an average of 44 iterations (episodes) over 10 experiments (as it was found that the average is almost the same for values greater than 10 experiments) using random starting positions and with different initial rule-bases, the robot succeeded in getting out safely. The number of actions modified during the learning process was on average 15 rules in the obstacle avoidance behaviour. This shows that the algorithm has optimised the rule-base. Thus, instead of having to fill $3^3 = 27$ rules, the robot has found that it only needed 15 rules to complete the mission resulting in an overall rule reduction. Although the experiment lasts for about 25 mins, most of the time is spent moving backward and forward as this takes a long time due to the low speed of the robot. The control cycle lasts for 200 ms using a 20 MHz MC68020 microprocessor. We have tried the robot without an *Experience Bank* and have found that the robot gives the same solution after 80 iterations. This justifies the idea of *Experience Bank* as, besides preserving the system experience, it also speeds up the GA search by starting the GA from the best point found in the space and not from a random point. In addition, during the 10 experiments the robot did not get stuck in a local minimum because of our use of the adaptive genetic parameters.

Table 1 shows the best learnt rule-base which produced the fastest time to get out of the maze and with least normalised steering and minimum absolute average deviation from the desired safe distance (embedded in the MF) and maximum speed. The robot response for this best learnt rule-base is shown in Fig. 9(b). For this learnt rule base the first 6 rules were randomly initialised with left velocity assigned to medium and right velocity assigned to medium and rules 7, 8, 9 and 10 were

Table 1
The best learnt rule base of the obstacle avoidance behaviour for Fig. 9(b)

LFS	MFS	RFS	Left velocity	Right velocity
Near	Near	Med	High	Very low
Near	Near	Far	High	Very low
Near	Med	Med	High	Low
Near	Med	Far	High	Very low
Near	Far	Med	High	Very low
Near	Far	Far	High	Low
Med	Near	Near	Very low	High
Med	Near	Med	High	Very low
Med	Med	Low	Very low	High
Med	Far	Low	Very low	High
Med	Far	Med	Very low	High
Far	Near	Near	Very low	High
Far	Near	Med	Very low	High
Far	Med	Near	Very low	High

assigned so that left velocity is medium and right velocity is very low and the rest of the rules were assigned so that the left velocity is medium and the right velocity is very low.

After the robot had successfully found the obstacle avoidance rules to navigate in geometrical structure shown in Fig. 9(b), we placed it in different starting positions to test the repeatability and stability for 10 experiments. The robot had shown to be robust and its path is repeatable as it followed the same path produced during learning cycle with an average deviation of 1.8 cm showing that the path is repeatable. The system is also stable as it didn't crash into the original obstacles. This implies that the robot had not learnt a specific path starting from a certain point. We have also conducted the same experiments with two robots (different kinematics and sensor positions). Each robot has produced an average of 15 rules over 10 experiments starting from different positions and with different initial rule-bases. The robot response using the best performing rule-base after learning is plotted in Fig. 9(c). It can be seen that they are almost the same. It is noted that each robot has developed a slightly different rule-base to compensate for the difference in characteristics between the robots. This demonstrates that different robots are able to adapt themselves to their environments. This supports the argument that it is better to learn online rather than in simulation where it is assumed that all the robots have the same characteristics (Table 1).

Fig. 10(a) introduces the problem tackled by Bonarini [4] and Leitch [24] which is the conventional corridor tracking problem and achieving a goal at the end of the corridor. We will compare the results first with Bonarini in which he placed a 60 cm wide robot in a 3 m wide corridor before moving it to 4 and 2 m wide corridors, then to the complicated corridor shown in Fig. 10(d). We have conducted our experiments with a 25 cm wide robot preserving the same ratios with our corridors as Bonarini, starting with 1.25 m and then moving to 1.67 and 0.83 m corridors to test the portability of the rule bases [2]. In these experiments we destroyed the rules in the obstacle avoidance behaviour by setting all the rule consequences to “go right”, thus causing the robot to collide with walls in the corridor.

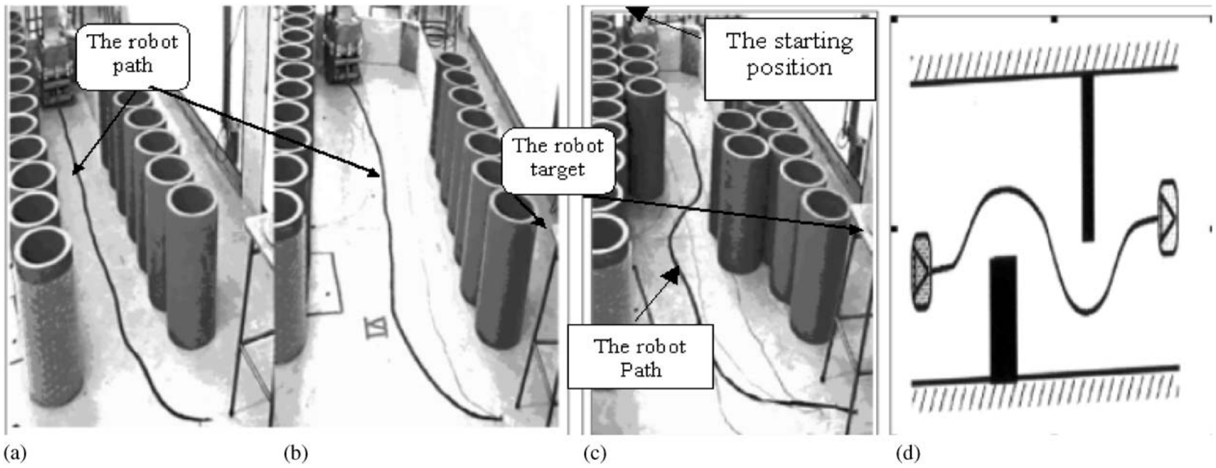


Fig. 10. Leitch and Bonarini corridor experiments: (a) Tight corridor. (b) Wide corridor. (c) Our algorithm. (d) Bonarini's method.

Table 2
The modified rules for the experiment in Fig. 10(c)

	LFS	MFS	RFS	Left speed	Right speed
Obstacle avoidance	Near	Far	Near	Med	Med
	Med	Near	Near	Very low	High
	Med	Med	Near	Low	High
	Med	Far	Near	Low	Med
	Far	Near	Near	Very low	High
	Far	Med	Near	Low	High
	Far	Far	Near	Very low	Med

We started the learning using the corridor shown in Fig. 10(c). It took the robot 16mins (including reversing time) to get out of the corridor and to modify the obstacle avoidance rule bases. It needed to modify only 7 rules in the obstacle avoidance behaviour. It learnt these rules in an average of 20 iterations (episodes) over 4 experiments, after learning it produced the path shown in Fig. 10(c). The modified rules are shown in Table 2. It is worth saying that in our episodes the number of control steps per episode are variable as the episode ends with the robots colliding or getting very close to an obstacle. The robot was started from 8 different positions in the corridor and followed the same path with a high degree of repeatability and stability. The robot did not crash at all (note the smooth response of the robot). The robot was then tried in the tight corridor and the wide corridor in Figs. 10(a) and (b) and the robot produced a smooth response and followed the centre-line of the corridors with an average deviation of 1.2 cm and a standard deviation of 0.7.

Leitch [24] only experimented with simple corridor-following in simulation using context dependent coding and succeeded in producing a solution after 40 generations (his rule-base would need

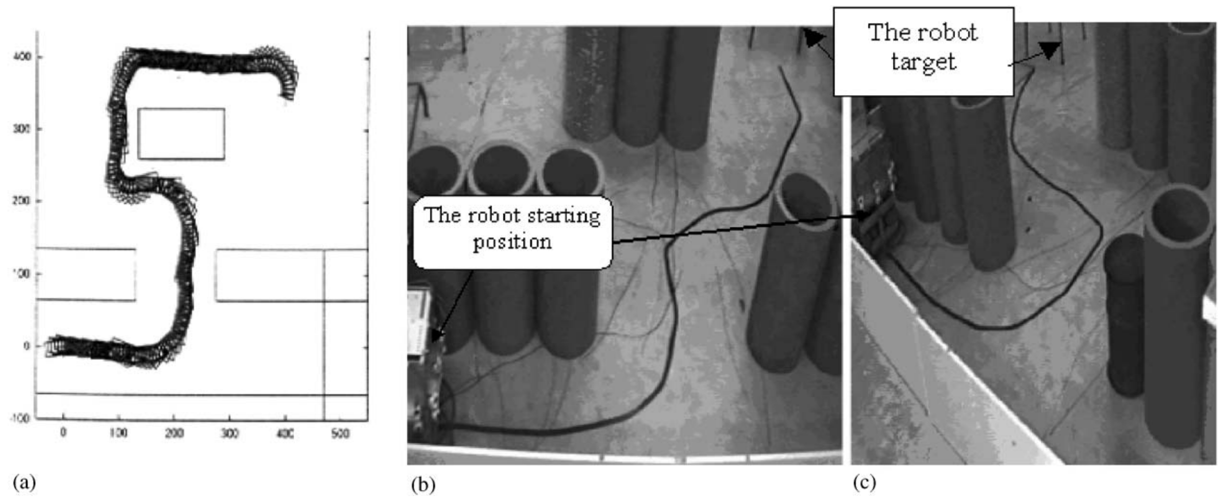


Fig. 11. Comparison to Hoffmann's work: (a) Hoffmann's method. (b) Our method with target to the right. (c) Our method with the target to the left.

Table 3
The learnt rule base for the obstacle avoidance behaviour in Fig. 11

	LFS	MFS	RFS	Left speed	Right speed
Obstacle avoidance	Near	Near	Med	High	Low
	Near	Near	Near	High	Vely low
	Med	Med	Far	High	Low
	Med	Med	Med	High	Med
	Med	Near	Near	Very low	High
	Far	Med	Med	Med	High

to be modified to solve the problem of Fig. 10(c). Bonarini used his algorithm with a simulated robot before transferring his controller to a real robot. To solve the problem in Fig. 10(d) Bonarini needed 471 leaning episodes (iterations) [4] while we needed only 20.

Hoffmann [17] introduced his method of incrementally tuning fuzzy controllers by means of an evolutionary strategy. In the benchmark problem of Fig. 11(a) where the target can be reached equally from left and right, so if the obstacle avoidance rules were learnt alone and not taking into consideration the other behaviours we will have the problem of conflicting outputs of the different behaviours explained in Section 2. Hoffmann has succeeded in finding a solution after 50 iterations with a rule base of 9 rules. In his previous work he used a messy-GA to learn the fuzzy controller. In this experiment the rule base was initialised so that all their actions go left. After 18 mins (including low robot speed and reversing) the robot successfully achieved its goal in an average of 21 iterations. It learnt 6 rules in the obstacle avoidance behaviour. As shown in the rule base shown in Table 3 the robot had learnt rules in the obstacle avoidance behaviour that when the obstacle can be avoided

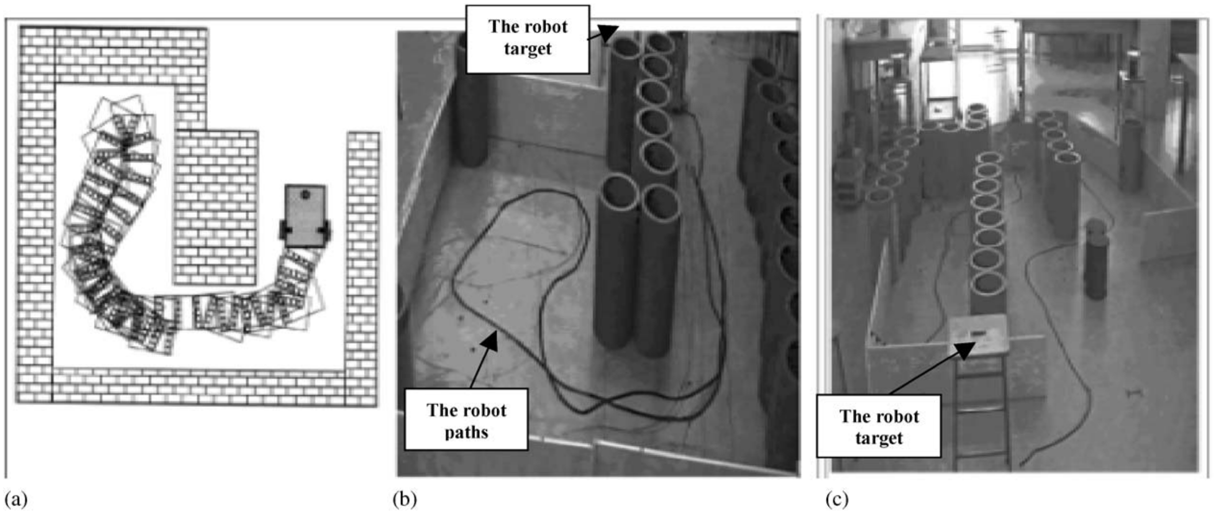


Fig. 12. Comparison to Hoffmann's work. (a) Hoffmann's method. (b) Our method. (c) The robot response starting from a different position.

from left or right i.e. when all sensors MF is medium or near, it favours turning to the right. This shows the importance of learning online using real robots rather than simulation as learning online enable the robots to choose rules that suits their capabilities. So when the robot is approaching the obstacle and it is far from the obstacle, the goal seeking behaviour that was learnt in our previous work [16] will influence more the motion of the robot that when the goal is aligned with the starting point of the robot it will go straight ahead till the obstacle approaches and the obstacle avoidance activation level increases as the robot approaches the obstacle and thus it influences more the robot motion turning it right to avoid the obstacle. This results in smooth response, also the robot path is robust as when the target is moved to the right the robot avoids the obstacle from the right as shown in Fig. 11(b) and when the target is placed to the left the robot avoids the obstacle from the left as shown in Fig. 11(c). These experiments proofs that our system can deal with problems of behaviour conflicting outputs as the robot learn online behaviour rules that takes into considerations its capabilities and the interaction with the other behaviours which are difficult for a human designer to estimate, thus leading to complementary behaviour outputs rather than contradictory outputs. The robot was subjected to 8 further trials to test its repeatability and stability. It followed the given path with high repeatability and stability.

The same controller from the previous experiment was applied to the problem in Fig. 12(a). In this experiment, the robot moves through a tight corridor into a wide area where it encounters a dead end before turning back to reach its goal outside the structure. We conducted this experiment with the previous controller to test its generality. The robot successfully completed the required task; the system response is shown in Fig. 12(b). This supports the generality of the learnt rules.

Fig. 12(c) shows the robot with the behaviours learnt from the previous experiment in Fig. 11 and applied to a different geometrical structure. Again the robot displays a good response escaping the structure and achieving its goal. In these experiments the robot showed a robust response reacting

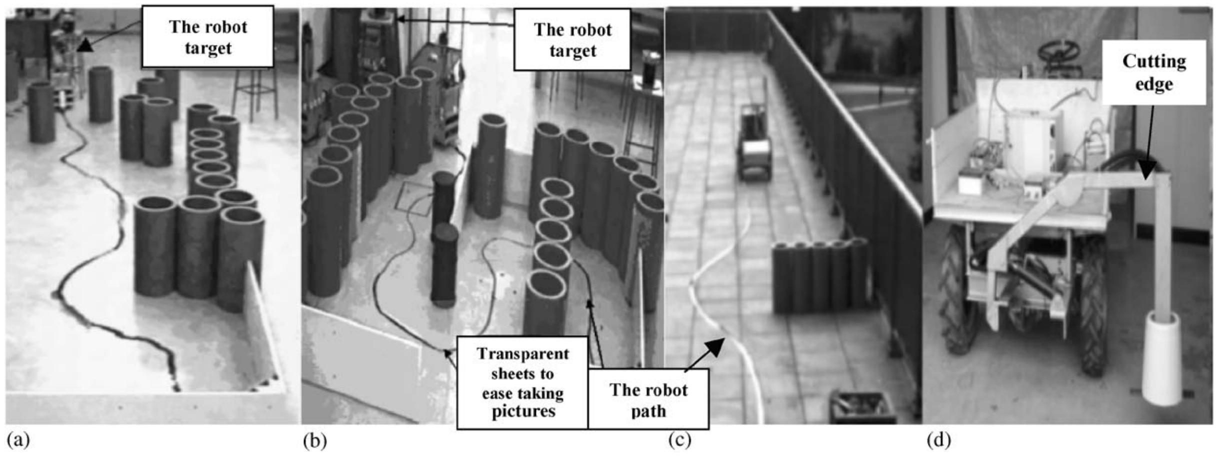


Fig. 13. (a) The robot after learning its controller using a bigger search space. (b) A robot with different kinematics adapting the robot controller learnt in Fig. 9(b). (c) The electrical outdoor robot learning the obstacle avoidance behaviour in outdoor environment. (d) The diesel tractor with its cutting edge.

to different geometrical structures different from the ones encountered during learning, which shows that our system had learnt general rule bases.

In the previous experiments in learning the obstacle avoidance behaviour rules online each sensor was represented by three MF and the output MF for the two actuators was represented by 4 MF. In order to demonstrate that our techniques can scale easily to big search spaces. We had performed the experiments in Fig. 13(a) in which we have tried first to increase the MF for the input sonar sensors to five and as we have three sensors this leads to a possible rule base of $5 * 5 * 5 = 125$ rules and we increased the output MF to 7. We were able to find a successful rule base after an average of 50 iterations over 5 experiments starting from different starting points and different initial rule bases. The robot learnt a rule base of 24 rules. We then increased the number of input MF to 7 for each input sensor this leads to a possible rule base of $7 * 7 * 7 = 343$ rules and we increased the number of the output MF to 9. We were able to find a successful rule base after an average 56 iterations over 5 experiments starting from different starting points and different initial rule bases. The robot learnt a rule base of 28 rules. Fig. 13(a) shows the robot response after learning this rule base. Although the response is smoother when increasing the MF but the path is almost the same as the smaller search spaces. These experiments had demonstrated that our system could still function as we scale up the search space.

In all the above experiments, we performed the statistical *t*-Test for matched (paired) samples over the worst and best rule-bases (in terms of maintaining the desired safe distance from obstacles). We found that the *t*-Test showed that the two solutions are statistically similar showing that our system can find a good solution and that the difference between the best and the worst rule-base is small. For example in Fig. 9(b), the best rule-base produced an absolute average deviation for the desired safe distance of 5 cm, while the worst rule-base produced an absolute average deviation of 8.5 cm. This is still much better than the manually designed rule-base which produced an absolute average deviation of 15 cm while using more rules. In addition, the learnt rule-base performed at faster speeds and used less steering deviation than the manually designed one.

In Fig. 13(b) we have coordinated the obstacle avoidance rule base learnt for the robot in Fig. 9(b) whose learnt rules are shown in Table 1 with the goal seeking and the left and right edge following behaviours that had been learnt in our previous work [11,12,16]. We have also learnt the best MF and coordination parameters for this HFLC in other work [14,15]. In order to proof that a short adaptation cycle can adapt a robot controller to another robot performing the same mission with different dynamics and kinematics. We have moved the learnt HFLC for the robot in Fig. 9(b) to another robot which has the same sensors configuration and the same actuators but it has different kinematics as it biased to the left so as the robot is applying steering by the difference in speed between the two wheels and as the robot dynamics are biased to the left this means that if both speeds are the same the robot will be drifted to the left.

In Fig. 13(b) the robot was given the high level mission of following the centre of the corridor while avoiding obstacles and achieving the goal. As the robot is operating with a controller that did not suit its capabilities it begun to collide with obstacles and deviate from the centre of the corridor (evaluated over a distance equivalent to twice the robot length as explained in Section 3.2.2). By applying our adaptation techniques explained in Section 3.2.2 the robot did not have to relearn the whole controller from the beginning, it just needed to tune different rules in different behaviours that if corrected will give the desired response. It took the robot 16 mins (including reversing time) to get out of the corridor and to modify the rule bases of the coordinated behaviours. It modified 8 rules in the obstacle avoidance behaviour, 4 rules in the left wall following behaviour, 3 rules in the right wall following behaviour and 3 rules in the goal seeking behaviour (i.e. 18 rule). It learnt these rules in an average of 20 iterations (episodes) over 4 experiments, and after learning it produced the path shown in Fig. 13(b). The modified rules are shown in Table 4. In the obstacle avoidance behaviour the system had introduced new rules not present for the controller developed in Table 1 which are *If LFS is Near and MFS is Far and RFS is Near then the Left speed is High and the Right speed is Med*, Human generated rules would have suggested that both wheel speeds be the same to pass between the closely located obstacles, but as the robot is biased to the left so turning it slightly to the right will help to compensate for the left bias and will cause the robot to go in a straight line. Also the robot had learnt a new rule which is *if LFS is Far and MFS is Far and RFS is Near* then robot turns sharply left to avoid collision. For the rest of the behaviours rules the system had tuned the rules that are affected by the bias so that the robot will give the desired response. This experiment proofs that our system is able to adapt a controller that was developed on a robot to another robot with the same sensor configuration doing the same mission in a relatively short time interval with no need to relearn the controller from the beginning. The system can add, delete and adapt rules according to its requirement. The robot was started from 8 different positions in the corridor and followed the same path with a high degree of repeatability and stability. The robot did not crash at all (note the smooth response of the robot). The robot produced a smooth response and followed the centre-line of the corridor with an average deviation of 1.2 cm and a standard deviation of 0.7 and was always able to achieve its goals when placed in different locations outside the corridor.

Fig. 13(c) shows the outdoor electrical robot learning the obstacle avoidance behaviour in outdoor unstructured environment. The robot converged to a solution after an average of 16 iterations taking 8 mins of robot time (this robot is faster than the indoor robot). The robot response is robust and can deal with different sizes of obstacles. We have experimented starting from different starting positions with different rule initial rule bases. Table 5 shows the best learnt rule

Table 4
The modified rules for the experiment in Fig. 13(b)

	LFS	MFS	RFS	Left speed	Right speed
Obstacle avoidance	Near	Near	Med	Med	Very low
	Near	Near	Far	Med	Very low
	Near	Med	Med	Med	Low
	Near	Med	Far	Med	Very low
	Near	Far	Med	Med	Very low
	Med	Near	Med	Med	Very low
	Near	Far	Near	High	Med
	Far	Far	Near	Very low	High
Left edge following	LSF	LSB	Left speed	Right speed	
	Med	Near	Low	Med	
	Far	Near	Very low	High	
	High	Med	Med	High	
	High	High	Med	High	
Right edge following	RSF	RSB	Left speed	Right speed	
	Near	Near	Very low	High	
	Low	Med	Low	High	
	Low	High	Very low	High	
Goal seeking	Bearing		Left speed	Right speed	
	Very very negative		Very low	High	
	Very negative		Med	High	
	Med negative		Med	Med	

Table 5
The learnt rule base for the electrical outdoor robot in Fig. 13(c)

LFS	MFS	RFS	Speed	Steering
Near	Near	Med	Low	High
Near	Near	Far	Low	High
Near	Med	Med	Med	High
Near	Far	Med	Med	Med
Near	Far	Far	High	High
Med	Near	Near	Low	Low
Med	Near	Med	Med	Very low
Med	Far	Near	Med	Med
Med	Far	Med	High	Med
Far	Near	Near	Med	Very low
Far	Med	Near	High	Low

base for the obstacle avoidance behaviour for the electrical wheel chair robot navigating in outdoor environments.

Table 6
The adapted rule base for the diesel tractor

	LFS	MFS	RFS	Speed	Steering
Obstacle avoidance	Near	Near	Med	Med	High
	Near	Near	Far	Med	High
	Near	Med	Med	High	High
	Near	Far	Med	High	Med
	Med	Near	Near	Med	low
	Med	Near	Med	High	Very low
	Med	Far	Near	High	Med
	Far	Near	Near	High	Very low
	Far	Med	Near	High	Low
Right edge following	RSF	RSB	Speed	Steering	
	Far	Near	Med	High	
	Far	Med	Med	High	
	Far	Far	High	High	
Goal seeking	Bearing		Speed	Steering	
	Very very negative		High	Very low	
	Very negative		High	Low	
	Med negative		Med	Low	

After learning the obstacle avoidance behaviour for the electrical outdoor robots, it was then co-ordinated this with other behaviours that receive immediate reinforcement, such as edge following and goal seeking. Using the techniques described in [12–16] the Membership Functions for each behaviour and the required coordination parameters were learnt in [14,15]. So we have learnt online a robot controller for outdoor navigation using a medium size electrical robot in a controlled environment. The same robot controller will be ported to another outdoor diesel tractor robot having the same sensor configuration as the outdoor electrical robot but with different shape and kinematics. The diesel robot actuators are pneumatic activated by the computer signals to control the speed and the steering of the robot. The diesel robot will perform the same function as the electrical robot of following a crop edge to cut it while avoiding obstacles and achieving a goal at the end of the edge which can be a station to empty the cultivated crop [12]. The robot and its cutting edge are shown in Fig. 13(d) So rather than having the diesel tractor perform the whole learning cycle online we have developed the robot controller on the electrical wheel chair robot and then moved it to the diesel tractor.

After about 19 iterations, the diesel robot had adapted the ported controller according to its objectives and its capabilities. As shown in Table 6 the robot had adjusted 9 rules in the obstacle avoidance behaviour and 3 rules in the right edge following behaviour and 3 rules in the goal seeking behaviour. It is worth noting that the robot did not activate the left edge following behaviour as it is not needed for its mission. As the diesel robot is slow then it had modified the rules consequents to go to a higher fuzzy output MF, this is obvious in the Obstacle avoidance behaviour where the robot had used the same steering values from Table 5 which had the learnt the obstacle avoidance

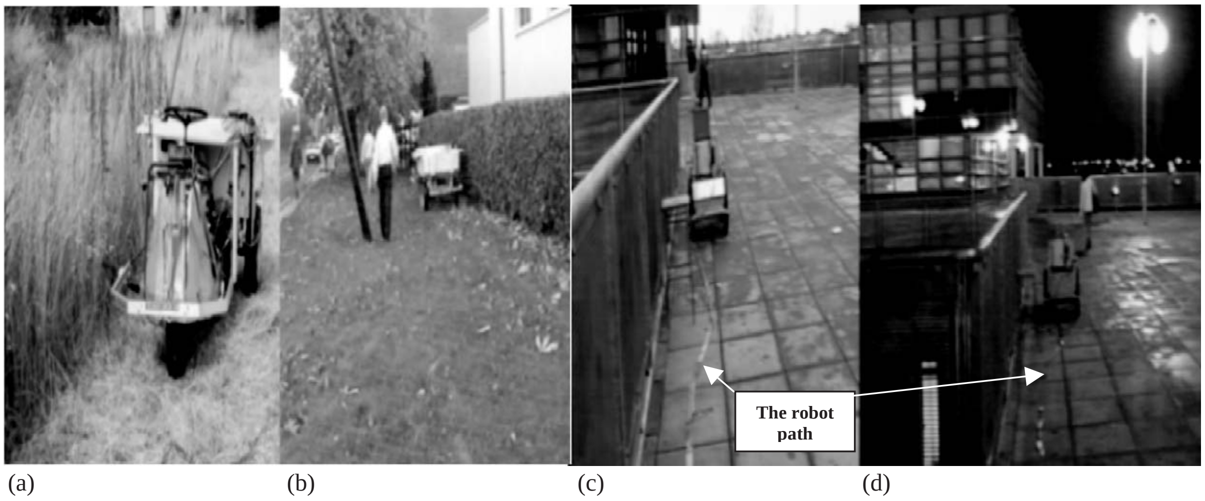


Fig. 14. (a) The diesel robot after adaptation follows an irregular hay crop. (b) The diesel robot after adaptation follows a tree hedge. (c) The robot performing life long learning during day time. (d) The robot performing life long learning during night-time.

behaviour for the electrical outdoor robot but it had raised the speed consequents to give a higher speed and thus improving the robot performance. The same applied for the goal seeking behaviour. In the right edge following behaviour when facing corners the electrical wheel chair actions were slight modification however in the diesel tractor when turning around a corner (*when RSF is far and RSB is Near or Med*) the robot should try maximum steering to move the heavy weight of the vehicle and achieve the desired response. In case the robot gets lost from the edge that is should follow to *cut* (*RSF is Far and RSB is FAR*) the robot will also apply maximum steering to recover back and follow the edge. We have tested the robot starting from different positions and following different edges as shown in Fig. 14(a) and (b) and the robot was always able to produce the same path and the robot response was robust as it reacted to various crop edges and dealt with moving object like humans and animals. The final result gave a small average deviation of 3 cm and a standard deviation of 2 for the edge following, and average deviation of 4 cm and standard deviation of 2 for the obstacle avoidance safe distance. While the robot was always able to achieve its goal placed at different positions.

These experiments shows that our algorithm can easily be moved between the prototype and the real vehicle requiring only minimal changes. This procedure is useful in learning controllers for vehicles which might involve dangerous maneuvers using small and cheap prototype robots in a controlled environment, thus avoiding the risks associated with online learning using heavy and expensive vehicles. This saves the need to generate a new controller for each different vehicle.

As we are involved in a big project concerning autonomous vehicle navigation in outdoor unstructured and dynamic environments [12]. One requirement is that the robot will implement a life long learning approach as it will be able to navigate autonomously in the outdoor agricultural environments unattended for long times. So the robot should have the ability to adapt its controller online in a short time interval with no human intervention to any environmental or robot kinematics changes it might encounter.

To show that our system is capable of implementing the life long learning strategy we have tried our electrical outdoor robot coordinating the left edge following and obstacle avoidance behaviour to follow an irregular square metallic edge in outdoor environment continuously for 4 h. The robot initial performance for the specified operation was a very good response giving an average deviation of 3 cm and a standard deviation of 0.9 for the edge following and an average deviation of 4 cm and standard deviation of 0.9 for the obstacle avoidance safe distance. The experiments ran continuously for 4 h during the day as shown in Fig. 14(c) and during the night as shown in Fig. 14(d) and during sunshine, rain, etc. As it rained, the ground became more slippery and the edge following rules had to be adjusted so that the robot will not slip. The robot had adapted all the steering consequents so that it does not take any harsh decisions so all the Very Low were changed to Low consequents so the robot did slight changes to keep the desired objective without slipping, also all the speed consequents were dropped to Low. For the Obstacle avoidance behaviour the same happened where all the Very Low actions in steering were replaced with Low and the speed of the robot was slowed down. These same actions were going to be taken by a human driver driving in rainy condition which shows that our techniques are close to the human way of thinking, learning and adaptation. After the rain had finished and the ground had dried a little bit, the robot speeded up but still the steering actions were the same as the ground was damp and it was night time. Although it is difficult to assess the robot performance for 4 h in outdoor open environment however we estimate from the average robot path that the robot had followed the same path with a small deviation. So our robot is implementing a life long learning strategy where it is acquiring information and experience along its life not only during the learning and training sessions. These online adaptation and lifelong experiments show that our technique is valuable for such an outdoor changing and dynamic environment like the agricultural environment.

It is worth saying that these experiments were performed using different indoor and outdoor robots with different sizes and kinematics operating in different environments. This had given us the opportunity to check the robustness of all the algorithm parameters such as the *Experience Bank*, ending criteria, mutation and crossover probabilities. As shown from the above experiments the system had performed robustly with these parameters using different robots operating in different indoor and outdoor environments which shows that our algorithm parameters have not been designed for a specific robot navigating in certain environment but they were general so that the algorithm can be applied to different robots operating in different environments.

5. Complexity of our method and comparison with other work

In this section we will just study the complexity of our method and compare it with other work.

In learning the rule bases the design of our system does not learn the whole controller on one go like the Pittsburgh approach but the learning cycle is sub-divided into local situations. This reduces the size of the model to be learnt. The accent on local models implies the possibility to learn by focusing on a small part of the search space at each step. The interaction among local models, due to the intersection of neighbouring fuzzy sets causes the local learning to reflect on global performance [3]. This action reduces the learning time as we apply the divide and conquer strategy instead of attacking the whole problem for a global solution as other researchers have done [26,48]. Also as the fuzzy classes overlap they produce desirable effects on the produced controller, such as robustness, and smoothness of action [2]. However in order to achieve a sub-optimal solution for the individual behaviour (a subset of the large search space) we next need to find the most suitable membership functions for the newly learnt rules, as explained in [15]. After finding a sub-optimal solution for each behaviour we can combine these behaviours and learn the best co-ordination parameters that will give a "good enough" solution for the large search space to satisfy a given mission or plan, as explained in [16]. After learning the robot controller if the robot or environment condition changes the robot does not to repeat a time consuming offline learning cycle like other researchers [26,48]. Instead the online adaptation technique modifies the poor rules in the relevant behaviours to adjust the robot to differing environmental and kinematics conditions. This is termed *life long learning*, where the robot can adapt itself to any new situation and can update its knowledge about its environment. This hierarchical approach enable us to learn online through interaction with the environment and using real robot controllers that are good enough to do the required mission.

In this paper we have concentrated on leaning the rule bases of behaviours that receive delayed reinforcement like the obstacle avoidance behaviour. We have learnt the rule base online and through interaction with the environment and using different robots with different shapes, sizes and kinematics and operating in indoor and in outdoor unstructured environment. Also our system was able to operate in real time implementing a life long learning approach in unstructured and changing environments. According to the author's knowledge our evolutionary techniques are the first evolutionary techniques that can operate a mobile robot and online learn its controller in real outdoor unstructured environment like the agricultural environment and adapt the robot online it to its changing environments.

For learning the obstacle avoidance behaviour the system operates through learning steps detailed in Section 3 and its flowchart is given in Fig. 4. These learning steps are simple and can be easily reproduced by other researchers as all the system parameters are related to the robot dimensions so that they can be applied easily to different robots. In order to validate our system we have learnt the robotic controller for different robots of different sizes and shapes and they all had given very good results as shown in the experiments section. Also our algorithm is not computationally expensive and can be used by vehicles with modest onboard computers (68040 with 4 Mbyte of RAM).

We do not claim that our system can find optimum controllers but they find controllers that are good enough to do the required job as in unstructured environments it is difficult to define what is optimum as what can be optimum in certain conditions will not be optimum if the environment changes which is a characteristic of unstructured environments such the outdoor agricultural environment. Also in outdoor navigation it is required to adapt the robot to any changes it might encounter with no need to repeat a time consuming cycle. The learnt controller is learnt online so it takes into consideration the sensors and actuators uncertainty and imprecision and it can adapt in relatively short time interval to any changes with no need to repeat a time consuming learning cycle.

Some researchers have tried to develop their controllers in simulation and then download them to real robots navigating in simple indoor environments. For example, Matellan et al. [26] have designed a GA to find the optimum obstacle avoidance rule base needed for indoor robot navigation. Their method optimised populations of rule-bases, rather than populations of rules as in our case. As their system tries to optimise the robot controller as a whole using a Pittsburgh approach to produce populations of rule bases rather population of rules as in our case. Their system takes a lot of iterations to converge to a solution which makes it not suitable for online learning. Moreover they test the system in simulation and then downloaded it to the real robots. As explained above in

Section 1.2 that using computer simulations for developing robot controllers has significant disadvantages which are best illustrated by the fact that when transferring the learnt controllers from the simulated world to the real world these controllers will usually fail [27]. Also if the robot has to operate in outdoor unstructured and dynamic environments and any robot or kinematics changes happen the system has to repeat a time consuming learning cycle which has to be repeated each time the environment changes.

Similarly Leitch [24] has used his genetic context dependent coding method to learn individual behaviour like edge following and obstacle avoidance. Again his algorithm evolved populations of rule bases offline and in simulation and it took long time to converge, while our system had developed complete controllers online and through interaction with the environment and was able to operate in life long learning mode where it adapted the robot to any changes with no need to repeat the whole learning cycle.

Hoffmann [17] introduced his method of incrementally tuning fuzzy controllers by means of an evolution strategy. In his previous work he used a messy-GA to learn the fuzzy controller. In his work he developed his robot controller in simulation and then transferred it to the real robots operating in indoor environments. It took him 50 generation $*$ 20 populations of controllers (trials) which needs 1000 iterations of the robot moving to test the solution. Again as explained above this will suffer from the problem associated with simulations. In the experiments section we have compared the robot response with his response in Fig. 11.

Bonarini [3] has produced an evolutionary technique for anytime learning and adaptation of structured fuzzy behaviours. In his technique he used his approach to adapt the behaviours learned in simulation to real environments and agents. It has a simulated model where his ELF learning takes place. The simulation model is updated by a monitor module, which continuously updates a set of tables representing the mapping from the control actions to their actual values. His techniques had been tested in indoor environments but not for outdoor changing and dynamic environments which we have achieved. In the experiments section we have compared our performance with Bonarini's ELF in Fig. 12.

There have been other work done in learning robot controllers not using evolutionary methods. Zhang [48] had produced a method for learning behaviours of mobile robots based on B-spline fuzzy controllers. He learnt online robot behaviours such as obstacle avoidance using a small Khepera robot. His system was only restricted to one output value (steering) and not general for multi-output system such as our system. His system took about 90 mins to learn the obstacle avoidance behaviour, while our system had taken shorter time to learn the same behaviour rule base. Also his system exhibits a problem where an obstacle could be avoided by passing to either side. His solution is to arbitrary choose which side to pass the obstacle, possibly causing conflicts when coordinated with other behaviours and leading to longer paths. The approach suggested in this paper not only converges more quickly, but also produces a solution that does not conflict with other behaviours.

Other researchers have also tried developing autonomous robots that learn the fuzzy controllers using reinforcement learning. Yung [47] needed 350 iterations to learn a controller in simulation. Faria [8] used a modified R-learning method to learn the obstacle avoidance behaviour which needed 10000 iterations in simulation. It can be seen that these times are very slow and thus cannot easily be applied to online learning.

Another approach is to use Neuro-fuzzy systems. Wang [45] has developed a self-adaptive Neuro-fuzzy system in which he used in simulation to develop a controller for underwater vehicles.

However, this requires training data, which will not be available in an online learning situation. In addition, Kasabov [21] has developed a system for evolving connectionist and fuzzy systems. He has successful applied this approach to the online learning of certain problems, such as speech recognition. However, because of the number of trials required before achieving a satisfactory result, it is not clear whether this approach would be appropriate to autonomous mobile robots.

Nehmzow [28] has carried out some work on life-long learning, but this was limited to building a map of an indoor environment.

In all the above methods none have tried his evolutionary algorithms to outdoor robots navigating in outdoor and unstructured environment like our case.

6. Conclusions and future work

In this paper, we have presented our novel Fuzzy–Genetic techniques for online learning, control and adaptation of an intelligent navigator for autonomous mobile robotic agents operating in unstructured and changing environments. These techniques are based on a newly patented double hierarchical Fuzzy–Genetic system which is able to learn and adapt the complex behaviours of intelligent robots in a short time interval without human intervention and implementing life long learning. We have focused in this paper on the learning of the obstacle avoidance behaviour, which is an example of behaviours receiving delayed reinforcement. The robot learns the obstacle-avoidance rule-base by first learning different situations (local solutions). Due to the intersection of neighbouring fuzzy sets, the local learning reflects on the global performance. This action reduces the learning time as we apply the divide and conquer strategy instead of attacking the whole problem for a global solution as other researchers have done. Also we use an *Experience Bank*, which besides preserving the system experience, also speeds up the GA search by starting the GA from the best found point in the space, not from a random point. In addition, we employ *Contextual Constraints* using the system's sensors so that the system will not have to search through the whole search space, but only in regions where the solutions are likely to be found. In addition, we used adaptive GA parameters in addition to simple but effective ending criteria. All of the algorithm parameters are robot size independent, not been designed for a specific kind of robot and thus are easily moved between differing robots. All of these techniques have resulted in fast convergence, learning the desired behaviour online via interaction with the environment in a relatively short time (bounded only by the robot speed). The learnt rule-base also has the ability to generalise when moved to other geometrical shapes not encountered during training. The system is flexible and can add rules or delete them in the case of changing environment or robot kinematics.

Evolving the robot controllers *online* enables the learnt controller to adjust to the real noise and imprecision associated with the sensors and actuators. By doing this we can develop rules that take such defects into account, producing a realistic controller for autonomous robotic agents, grounded in the physical world that emerge from strong coupling of the robotic agent and its environment not in simulation. These robotic agents are grounded in the real world (situated, embodied and operating in real time), as adaptive behaviours cannot be considered as a product of an agent in isolation from the world.

After developing a good controller in either the real robot or a prototype, the controller can then be moved to the target robot operating in a different environment. The target robot utilises a fast adaptation procedure to modify the learnt controller to the changing kinematics and environment. The principal advantage of using a prototype robot is that it avoids the problems associated with software simulation. It allows controllers to be learnt using small and cheap prototype robots in a controlled environment, thus avoiding the risks associated with online learning using heavy and expensive robots. Our life-long learning scheme, where the robot gains experience through its lifetime, helps to continually refine the controller without the need for retraining or human intervention. Our experiments have shown that over 4 h of continuous operation the robot was able to navigate continuously in rainy conditions adapting its controller and maintaining almost its path irrespective of changes in the ground conditions and robot kinematics. The robot had implemented a life long learning technique thus stepping beyond being a robot with a pre programmed controller or a controller that had been learnt under static conditions and that needs relearning from the beginning if any change occurs.

Our techniques had been verified in difficult domains which are difficult to solve using the current learning and adaptation techniques, such as robots navigating in unstructured environments (in particular the agricultural environment).

Advancing the state of knowledge in the field of online learning has potential benefits for a wide set of embedded control systems such as vehicles, factory machinery, telecommunication medical instrumentation and emerging areas such as intelligent-buildings and flying robots. It is also particularly appropriate to situations where modelling or reprogramming are difficult or costly (e.g. inaccessible environments such as underwater, outer space or environments where one agent is required to accomplish a variety of tasks). In such environments it is necessary to perform rapid online learning through interaction with the real physical world. Such an approach both saves money and increases reliability by allowing the agent to automatically adapt without further programming to the changing user and environment needs it will experience throughout its lifetime.

References

[1] H. Beom, H. Cho, A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning, IEEE Trans. Systems Man Cybernet. 25 (3) (1995) 464–477.

[2] A. Bonarini, Delayed reinforcement, fuzzy *Q*-learning and fuzzy logic controllers, in: F. Herrera, J.L. Verdegay (Eds.), Genetic Algorithms and Soft Computing, (Studies in Fuzziness, 8), Physica-Verlag, Berlin, D, 1996, pp. 447–466.

[3] A. Bonarini, Anytime learning and adaptation of hierarchical fuzzy logic behaviours, Adap. Behav. J. (Special Issue on Complete Agent Learning in Complex Environments) M. Mataric (Ed.) 5 (1997) 281–315.

[4] A. Bonarini, F. Basso, Learning to co-ordinate fuzzy behaviours for autonomous agents, Internat. J. Approx. Reason., special issue on genetic-fuzzy systems for control and robotics 17 (4) (1997) 409–432.

[5] R. Brooks, Artificial Life and Real Robots, MIT Press, Cambridge, 1992.

[6] M. Delgado, F. Zuben, F. Gomide, Evolutionary design of Takagi–Sugeno fuzzy systems: a modular and hierarchical approach, Proc. the 2000 IEEE Int. Conf. on Fuzzy Systems, San Antonio-USA, 2000, pp. 447–452.

[7] M. Dorigo, M. Colombetti, Robot shaping: developing autonomous agents through learning, Artificial Intelligence J. 71 (1995) 321–370.

[8] G. Faria, R. Romero, Incorporating fuzzy logic to reinforcement learning, Proc. 2000 Int. Conf. on Fuzzy Systems, San Antonio-USA, 2000, pp. 847–851.

[9] T. Fukuda, N. Kubota, An intelligent robotic system based on fuzzy approach, Proc. IEEE 87 (9) (1999) 1448–1470.

[10] D. Goldberg, Genetic Algorithms in Search, Optimisation and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[11] S. Goodridge, M. Kay, R. Luo, Multi-layered fuzzy behaviour fusion for reactive control of an autonomous mobile robot, Proc. 6th IEEE Int. Conf. on Fuzzy Systems, Spain, 1997, pp. 573–578.
[12] H. Hagras, V. Callaghan, M. Colley, A fuzzy-genetic based embedded-agent approach to learning and control in agricultural autonomous vehicles, Proc. 1999 IEEE Int. Conf. on Robotics and Automation, Detroit-U.S.A, 1999, pp. 1005–1010.
[13] H. Hagras, V. Callaghan, M. Colley, Online learning of fuzzy behaviours using genetic algorithms & real-time interaction with the environment, Proc. 1999 IEEE Int. Conf. on Fuzzy Systems, Seoul-Korea, 1999, pp. 668–672.
[14] H. Hagras, V. Callaghan, M. Colley, On line calibration of the sensors fuzzy membership functions in autonomous mobile robots, Proc. 2000 IEEE Int. Conf. on Robotics and Automation, San Francisco-USA, 2000, pp. 3233–3238.
[15] H. Hagras, V. Callaghan, M. Colley, Learning fuzzy behaviour co-ordination for autonomous multi-agents online using genetic algorithms & real-time interaction with the environment, Proc. 2000 IEEE Int. Conf. on Fuzzy Systems, San Antonio-USA, 2000, pp. 853–859.
[16] H. Hagras, V. Callaghan, M. Colley, Prototyping design and learning in outdoor mobile robots operating in unstructured outdoor environments, IEEE Int. Mag. Robot. Automat. 8 (3) (2001) 53–69.
[17] F. Hoffmann, Incremental tuning of fuzzy controllers by means of evolution strategy, Proc. GP-98 Conf., Madison, Wisconsin, 1998, pp. 550–556.
[18] J. Juan, Implementation of path planning on an autonomous mobile robot, M.sc Thesis, University of Essex, 1999.
[19] L. Kaelbing, M. Littman, Reinforcement learning: a survey, J. Artif. Int. Res. 4 (1996) 237–285.
[20] N. Kasabov, Introduction: hybrid intelligent adaptive systems, Int. J. Intelligent Systems 6 (1998) 453–454.
[21] N. Kasabov, M. Watts, Neuro-genetic information processing for optimisation and adaptation in intelligent systems, in: N. Kasabov, R. Kozma (Eds.), Neuro-Fuzzy Techniques for Intelligent Information Processing, Physica-Verlag, Heidelberg, 1999, pp. 97–110.
[22] C. Lee, Fuzzy logic in control systems: fuzzy logic controller, Part I, IEEE Trans. Systems Man Cybernet. 20 (1990) 404–432.
[23] C. Lee, Fuzzy logic in control systems: fuzzy logic controller, Part II, IEEE Trans. Systems Man Cybernet. 20 (1990) 419–434.
[24] D. Leitch, A new genetic algorithm for the evolution of fuzzy system, Ph.D. Thesis, University of Oxford, 1995.
[25] G. Linkens, O. Nyongeso, Genetic algorithms for fuzzy control, Part II: online system development and application, IEE Proc. Control Theory Appl. 142 (1995) 177–185.
[26] V. Matellan, C. Fernandez, J. Molina, Genetic learning for fuzzy reactive controllers, J. Robot. Autonomous Systems 25 (1998) 33–41.
[27] O. Miglino, H. Lund, S. Nolfi, Evolving mobile robots in simulated and real environments, Technical Report NSAL-95007, Reparto di sistemi Neurali e vita Artificale, Instituto di Psicologia, Consiglio Nazionale delle Ricerche, Roma, 1995.
[28] U. Nehmzow, Continuous operation and perpetual learning in mobile robots, Proc. Int. Workshop on Recent Advances in Mobile Robots, Leicester-UK, 2000, pp. 20–27.
[29] F. Pin, Y. Watanabe, Navigation of mobile robots using a fuzzy behaviourist approach and custom designed fuzzy inference boards, Robotica 12 (1994) 491–503.
[30] R.G. Reynolds, Chung, Chan-Jin, The use of cultural algorithms to support self-adaptation in evolutionary programming, Proc. 1996 Adaptive Distributive Parallel Computing Symp., Dayton-Ohio, 1996, pp. 260–271.
[31] R.G. Reynolds, Sternberg, Michael, Using cultural algorithms to support the re-engineering of rule-based expert systems in dynamic performance environments: a fraud detection example, IEEE Trans. Evolutionary Computation 1 (4) (1997) 225–243.
[32] M. Rocha, P. Cortez, J. Neves, The relationship between learning and evolution in static and dynamic environments, Proc. Second ICSC Symp. on Engineering of Intelligent Systems, Paisley, UK, 2000, pp. 500–506.
[33] A. Saffiotti, Fuzzy logic in autonomous robotics: behaviour co-ordination, Proc. 6th IEEE Int. Conf. on Fuzzy Systems, Barcelona, Spain, 1997, pp. 573–578.
[34] D.L. Schacter, Implicit memory: history and current status, J. Exper. Psychol. Learning, Memory, Cogn. 13 (1987) 501–518.
[35] A. Schwartz, A reinforcement learning method for maximising undercounted rewards, Proc. 10th Internat. Conf. Machine Learning, Morgan Kaufmann, San Mateo-CA, 1993, pp. 450–457.

[36] R. Shanks, Human instrumental learning: a critical review of data and theory, British J. Psychology 84 (1993) 319–354.
[37] Y. Shi, R. Eberhart, Y. Chen, Implementation of evolutionary fuzzy systems, IEEE Trans. on Fuzzy Systems 7 (2) (1999) 109–119.
[38] M. Sousa, M. Madrid, Optimisation of Takagi–Sugeno fuzzy controllers using a genetic algorithm, Proc. 2000 IEEE Int. Conf. on Fuzzy Systems, San Antonio-USA, 2000, pp. 30–36.
[39] W. Spears. Crossover or mutation? Proc. Foundations of Genetic Algorithms Workshop, 1992, pp. 221–237.
[40] M. Srinivas, L. Patnaik, Adaptation in genetic algorithms, in: S. Pal, P. Wang (Eds.), Genetic Algorithms for Pattern Recognition, CRC Press, Boca Raton, FL, 1996, pp. 45–64.
[41] J. Sushil, J. Johnson, Solving similar problems using genetic algorithms and case-based memory Proc. Int. Conf. on Genetic Algorithms, Morgan Kauffman, San Mateo-CA, 1997, pp. 908–915.
[42] J. Sushil, J. Louis, L. Gan, Combining robot control strategies using genetic algorithms with memory, Lecture Notes in Computer Science, Evolutionary Programming, Vol. 1, Springer, Berlin, 1997, pp. 21–33.
[43] G. Tan, X. Hu, More on designing fuzzy controllers using genetic algorithm: guided constrained optimisation, Proc. 6th IEEE Int. Conf. Fuzzy systems, Barcelona-Spain, 1997, pp. 497–502.
[44] E. Tunstel, T. Lippincott, M. Jamshidi, Behavior hierarchy for autonomous mobile robots: fuzzy behavior modulation and evolution, Internat. J. Intelligent Automat. Soft Computing 3 (1) (1997) 37–49.
[45] J. Wang, J. Yuh, Self-adaptive neuro-fuzzy systems with fast parameter learning for autonomous underwater vehicle control, Proc. 2000 IEEE Int. Conf. on Robotics and Automation, San Francisco-USA, 2000, pp. 3861–3866.
[46] J. Yen, N. Pfluger, A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation, IEEE Trans. on Systems Man Cybernet. 25 (6) (1995) 971–978.
[47] N. Yung, C. Ye, An intelligent mobile vehicle navigator based on fuzzy logic and reinforcement learning, IEEE Trans. Systems Man Cybernet. 29 (2) (1999) 314–321.
[48] L. Zhang, V. Schwert, Rapid learning of sensor-based behaviours of mobile robots based on B-spline fuzzy controllers, Proc. 1998 IEEE Int. Conf. on Fuzzy Systems, Anchorage-U.S.A, 1998, pp. 1346–1352.
[49] H. Zhou, A computational model of cumulative learning, Mach. Learning J. 5 (4) (1990) 383–406.