In Journal of Fuzzy Sets and Systems, Volume 141, No. 1, pp. 107-160, January 2004

Learning and Adaptation of intelligent navigator in Autonomous Mobile Robots using a novel Fuzzy-Genetic System and Online Interaction with the Environment

Hani Hagras, Victor Callaghan, Martin Colley

Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, England, UK
Email: hakhag@essex.ac.uk

Keywords: Fuzzy Logic, Genetic Algorithms, Reinforcement Learning, Mobile Robots

Abstract

In this paper we will present our newly patented Fuzzy-Genetic techniques used to online learn and adapt an intelligent navigator for autonomous mobile robots navigating in unstructured and changing environments. In this work we will focus on online learning of the obstacle avoidance behaviour which is an example of behaviours that receive delayed reinforcement. The obstacle avoidance behaviour is then co-ordinated with behaviours that receive immediate reinforcement (such as goal seeking and edge following) learnt in our previous work to generate an intelligent reactive navigator. Our system is using a life long learning paradigm where it adapts to new environments and updates its knowledge, this technique gives the robot the ability to navigate in changing outdoor environments where it adapts itself with no need to repeat the learning cycle.

1- Introduction

In this work we will focus on learning the obstacle avoidance behaviour which is an example of behaviours that receive delayed reinforcement in autonomous intelligent mobile robotic agents. These are behaviours when reinforcement is not given at each control cycle, but only at a later time (*delayed reinforcement)*. The obstacle avoidance behaviour is then co-ordinated with other behaviours that receive immediate reinforcement (such as goal seeking and edge following) learnt in our previous work [11, 12, 13, 14, 15, 16] to generate an intelligent reactive navigator. This navigator then navigates in its changing and unstructured environment and implements a life long learning paradigm where it acquires experience and updates its knowledge and adapts to the new environments. Our technique gives the robot the ability to navigate in changing outdoor environments like the agricultural environment where it adapts itself with no need to repeat the learning cycle.

In this work we target autonomous mobile robotic agents. These agents aim to build physical systems that can accomplish useful tasks without human intervention in the real-world operating in unmodified environments i.e. in environments that have not been specifically engineered for the agent. To accomplish a given task, an agent collects or receives sensory information concerning its external environment and takes actions within the dynamically changing environment. However, the control rules are often dictated by human operators, but ideally the agent should automatically perform the tasks without assistance. Consequently the agent must be able to perceive the environment, make decisions, represent sensed data, acquire knowledge, and infer rules concerning the environment. Agents that can acquire and usefully apply knowledge or skill are often called Intelligent, these agents receive a task from a human operator and must accomplish the task in the available workspace [10].

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 that the agent should operate in. 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. 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 of autonomous robotic agents operating in changing unstructured environments is the large amounts of uncertainty that characterises 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 apriori, passageways may be blocked, and so on. On the other hand, knowledge acquired through sensing is affected by uncertainty and imprecision. The quality of sensor information is influenced by sensor noise, the limited field of view, the conditions of observation, and the inherent difficulty of the perceptual interpretation process.

Reinforcement Learning (RL) requiring only a scalar reinforcement signal as a performance feedback from the environment seems to be quite attractive when navigating in dynamic unknown environments. Reinforcement signal enables the navigator to tune its performance to some extent. RL is defined not by characterising a learning problem, any algorithm that is well suited for solving that problem is considered a reinforcement learning problem [9]. There are two main strategies for solving reinforcement learning problems. The first is to search in the space of behaviours in order to find one that performs well in the environment. This approach has been taken by work in genetic algorithms and genetic programming which is will be our case in this paper. The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world [20], this approach had been used to learn fuzzy systems for mobile robots as in [9], [43], [1]. However the statistical techniques such as Q learning and R learning have theoretically limited learning ability as it requires heavy learning phases and in some cases, it might not be able to completely capture the complex features of an environment [20].

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 or flying robots (helicopters) 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 in a short time interval. In our work we will implement online learning and we will 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 the remaining subsections of Section (1) we are going to explain what is meant by intelligent autonomous robotic agents and then we are going to explain why we aim to learn online using real robots with no need for simulation. Then we will introduce reinforcement learning and when delayed reinforcement is required. Then will introduce some methods used to develop robotic agents. In Section (2) we will introduce the hierarchical fuzzy logic controllers. In Section (3) we will introduce our hierarchical fuzzy genetic systems for online learning and adaptation and how to use it to learn the obstacle avoidance behaviour. In Section (4) we will introduce our online adaptation technique and the life long learning strategy. In Section (5) we are going to present our experimental results. In Section (6) we are going to introduce other related work and how our work over performs the current techniques. In Section (7) we present conclusions and future work.

1.1 Intelligent Autonomous Embedded Agents

According to Steels [38] an agent is a system that is capable of maintaining itself. An agent therefore must worry about two things: (i) performing the function (or set of functions) that determines its role in larger units and (ii) maintaining its own viability. This definition of an agent is so far almost completely identical with that of a living system as the living systems continuously replace their components and that way secure existence in the face of unreliable or short lived components. Also these systems as a whole adapt/evolve to remain viable even if their environment changes.

The central idea in the concept of autonomy is identified in the etymology of the term: *autos* (self) and *nomos* (rule or law). It was first applied to the Greek city states whose citizens made their own laws, as opposed to living according to those of an external governing power [38]. To be autonomous you must first be automatic. This means that you must be able to operate in an environment, sense this environment and impact it in ways that are beneficial to yourself and to tasks that are crucial to your further existence. But autonomy goes beyond automaticity, because it also supposes that the basis of self-steering originates (at least partly) in the agent's own capacity to form and adapt its principles of behaviour. Moreover the process of building up or updating competence is something that takes place, while the agent is operating in the environment (not in simulation). It is not the case that the agent has the time to study a large number of examples or to think deeply about how it could cope with unforeseen circumstances. Instead, it must continuously act and respond in order to survive. AI systems built using the classical approach are not autonomous, although they are automatic. Knowledge has been extracted from experts and put into the system explicitly. It is not done by the system itself. Current robotic systems are also automatic but so far not autonomous. These systems can never step outside the boundaries of what foreseen by the designers because they cannot change their own behaviour in a fundamental way. Biological systems are autonomous. Their structure is not built up by an outside agency, but they develop and maintain their internal structure and functioning through mechanisms like self-organisation, evolution, adaptation, and learning; and they do so while remaining viable in the environments in which they operate [38].

According to Kasabov [21] an Intelligent Agent System (IAS) should be able to learn quickly from large amounts of data. He also sates that an intelligent system should also adapt in real time and in an on-line mode as new data is encountered. Also the system should 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 and intelligent buildings had satisfied these criteria.

In summary we can say that robotic agents are self-sustaining systems which perform a function for others and thus get the resources to maintain themselves. But because they have to worry about their own survival they need to be autonomous, both in the sense of self-governing and of having their own motivations. Because environments and users of systems continuously change, the robotic agents have to be adaptive. Intelligence helps because it gives systems the capacity to adapt more rapidly to environmental changes or to handle much more complex functions.

1.2 Why Online Learning

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 [Dorigo 95]. Despite this, many embedded agents 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 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 [29]:

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 behaviors, these behaviors if learnt successfully are usually frozen to the robot. Thus if some of the agent dynamics or the environmental circumstances are changed, the robot must repeat a time-consuming learning cycle [29]. From the above discussion it is clear that using computer simulations for developing robot controllers has significant disadvantages which is best illustrated by the fact that when transferring the learnt controllers from the simulated world to the real world these controllers will usually fail [29]

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. Learning 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 takes such defects into account, producing a realistic controller for autonomous robotic agents, grounded in the physical world that emerge from strong coupling of the agent and its environment not in simulation. These 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.

1.3 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. The process of finding such path is also known as the path planning problem which could be classified into: global path planning and local path planning. Global path planning methods are usually conducted off-line in a completely known environment [43]. In the global path planning methods, an exact environment model has been used in planning the path. In these methods, the time complexity grows with the geometry complexity and grows exponentially with the number of degrees of freedom in the vehicle's motion. Thus they are only practical when the environmental model is simple and the number of degrees of freedom is reasonably low. However real environments are never simple enough. On the other hand, the local path planning techniques, also known as the obstacle avoidance methods, are potentially more efficient in vehicle navigation when the environment is unknown (which is our case) or only partially known [43]. An efficient local path planning method is the potential field method, which was first proposed by Khatib [23] and has been widely used in obstacle avoidance [4]. In spite of its simplicity and elegance, this method has three problems: First, local minimum could occur and cause the vehicle to be stuck; second it cause unstable motion in the presence of obstacles; and third it is difficult to find the force coefficients influencing the vehicle's velocity and direction in unknown environments [43]. The practicality of the potential field therefore hinges on how well these problems can be resolved. One of the possible solutions that has the potential to overcome these problems is the reactive system proposed by Brooks [5]. The key idea is to build a mapping from the perceived situations to the correct actions and iterate the mapping until a goal is reached. As previous reactive systems are not able to acquire situation-action rules, neural networks and fuzzy logic approaches offer an attractive alternative for building reactive systems in recent times [40].

Neural networks had been used for embedded agents development namely navigation of mobile robots [31]. Neural Networks have the ability to learn by example and generalise their behaviour to new data, however this supervised learning methods require substantially large set of representative patterns to characterise the environment during training, it is also difficult to obtain the training patterns which contain no contradictory input output pairs. It is also well known that these networks learn rather slowly in high dimensional spaces (which includes robots operating in unstructured environments). Also the network structure, i.e. the number of hidden units, needs to be chosen carefully to achieve good learning results. Too many nodes in the hidden layers of the network may cause the network to overfit the training data, while too few may reduce the its ability to generalise. Selection of the optimal number of layers and nodes is a difficult problem that is widely solved by a 'rule of thumb' approach, which is directly proportional to the experience of the network designer [22]. Since an autonomous mobile robotic agent in a changing unstructured environment will encounter new data all the time, complex off-line retraining procedures would need to be devised and substantial amount of data would have to be stored for the retraining, from a practical point of view this approach is hardly useful [41].

Fuzzy sets and systems constitute one of the most fundamental and influential computational intelligence tools [40]. 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 success of Fuzzy Logic Control (FLC) is owed in a large part to the technology's ability to convert qualitative linguistic descriptions into complex mathematical functions and the ability to deal with various situations without analytical model of the environment. The methodology of the FLC appears very useful when the processes are too complex for analysis by conventional quantitative techniques or when the available sources of information are interpreted qualitatively, inexactly or uncertainly, which is the case of autonomous mobile robots navigating in unstructured environments. Fuzzy logic approach seems quite promising in tackling the problem of obstacle avoidance, as it deals with various situations without requiring to construct an analytical model of the environment. While compared with the neural network approach, it has another distinct advantage that each rule of the rule base has a physical meaning. This makes it possible to tune the rules by using expert's knowledge. However in complex unstructured environments the necessity of a high quality information, to be extracted from a broad knowledge domain, constrains the application of fuzzy systems to solve very demanding problems. 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 embedded agents) 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 [43]. That is why automatic design of fuzzy systems represents a very promising and challenging research area [7] 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 posses the following advantages, they are efficient, possessing the ability to solve certain combinatorial problems many orders of magnitude faster than iterative searches. They 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. They are implicitly parallel where the solution is explored in parallel by searching different regions, this characteristic allows a global search in the solution space [22]. GA approach enriches the optimization environment for fuzzy systems. As described in [7], developing an optimal fuzzy system is equivalent to finding the minimum of a hyper-surface associated with an objective function. The hyper-surface has the following characteristics: it is infinitely large, complex and noisy; it is non-differentiable, since changes in the number of fuzzy rules are discrete and can have a discontinuous effect on the fuzzy system's performance; it is multi-modal (different fuzzy rule sets and/or membership functions may have similar performance) and deceptive, since a little modification may cause huge effects on the performance of each system [7]. There is much work reported in the literature on designing fuzzy controllers using GA [8, 29, 27, 26, 39, 19, 28, 3, 10, 7, 36]. 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 (i.e. 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 [21]. Hence prior to our work it was *not considered feasible for a simple GA to online learn and adapt a robotic controller* [27] in outdoor unstructured environments.

1.3 Reinforcement Learning

Reinforcement Learning (RL) is the problem faced by 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 that problem is considered a reinforcement learning problem [9]. There are two main strategies for solving reinforcement learning problems. The first is to search in the space of behaviours in order to find one that performs well in the environment. This approach has been taken by work in genetic algorithms and genetic programming which will be our case in this paper. The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world [20], this approach had been used to learn fuzzy systems for mobile robots as in [9], [43], [1]. RL differs from the more widely studied problem of supervised learning in several ways. The most important difference is that there is no presentation of input/output pairs. Instead after choosing an action the agent is told the immediate reward and the subsequent state, but not told which action would have been in its best long-term interests. It is necessary for the agent to gather useful experience about the possible system states, actions, transitions and rewards actively to act optimally. Another difference from supervised learning is that on-line performance is important: the evaluation of the system is often concurrent with learning (which seems attractive in online learning in unstructured environments). In the standard reinforcement learning model, an agent is connected to its environment via perception and action. On each step of interaction the agent receives input indicating the current state of the environment, the agent then chooses an action to generate an output. The action changes the state of the environment and the value of this state transition is communicated to the agent through a scalar reinforcement signal, the agent's behaviour should choose actions that tend to increase the long-run sum of values of the reinforcement signal. However 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 complex features of complex environments such as outdoor unstructured environments. To obtain the necessary information for learning, the agent explores the environment through the combined use of trials and errors and delayed rewards. This form of exploration searches for direction information on the basis of the environment properties. In doing so the reinforcement learning is slowed down.

This phenomena is known as conflict between exploration and exploitation which causes slow and uncertain convergence and insufficiently learnt rule base in most cases [43]. 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 [20].

Q learning is the most popular and seems to be the most effective model-free algorithm for learning from delayed reinforcement, it updates the expected discounted reinforcement by taking action a in state s. Since Q learning discounts future rewards, it prefers actions that result in short-term perhaps mediocre reward to those that result in long-term sustained rewards. 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 problems [9]. Shwartz [35] examined the problem of adapting Q learning to an average frame and he called it R learning. In R learning the agent is supposed to take actions that optimise its long-run average reward. Several researchers have found the average reward criterion closer to the true problem than the discounted criterion and therefore prefer R learning to Q learning [20]. R learning seems to exhibit convergence problems and it is quite sensitive to the choice of exploration strategy which makes it difficult for it to generalise to different environments [20].

All the previous methods has tacitly assumed that it is possible to enumerate the state and action spaces and store tables of values over them, this can be possible only in very small environments, this means impractical memory requirements. It also makes inefficient use of experience. In a large smooth state space we generally expect similar states to have similar values and similar optimal actions. Surely, therefore, there should be some more compact representation than a table. Most problems will have continuous or large discrete state spaces; some will have large or continuous action spaces.

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 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 make 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. For example, in case of the human trainers that try to guide by a joy-stick a robot to reach its prey, the action done at a given moment is not the best one, due to the imprecision of the control system, and of the sensorial feed-back to the trainer. However, the trainer is usually able to produce behaviours, whose effectiveness can be positively evaluated in few control actions.

Also some systems need time to express behaviours. Which is the case for the obstacle avoidance behaviour. For example, a robot blocked in a corner that should maneuver to escape. It should apply the obstacle avoidance behavior 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 behavior. The only possibility is to evaluate the robot's performance when it succeeds in escaping, and not 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 have eight inputs (7 sonar inputs and a bearing sensor) and two 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. If, as before, these behaviours represent each input by three fuzzy sets then obstacle avoidance with three inputs produces $3^3 = 27$ rules. The left edge following has two inputs producing $3^2 = 9$ rules, right edge following is the same and goal seeking has one input (more accurately represented by seven fuzzy sets) producing 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 co-ordination scheme is required to combine these behaviours into a single action. Saffiotti [34] 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.

We will use a method variant to the methods suggested by Saffiotti [34] and Tunstel el al [40] to apply fuzzy logic to both implement the individual behaviour elements and the necessary arbitration (allowing both fixed and dynamic arbitration policies to be implemented) [11]. 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 jobs (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. The plan can also define the situations in which the individual behaviours are active a process, which is more commonly referred to as “willed” operation. Doing this will produce a system that is capable of completing complicated tasks via co-ordinating a set of simple tasks which can be more easily designed. 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 and thus we can 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 [34]. As mentioned earlier, using a hierarchical strategy results in much fewer rules (i.e. a much simplified design problem) [34]. In addition, it allows flexible design where new behaviours can be added easily. Also, the system is capable of performing completely different tasks by the same basic behaviours just by changing the co-ordination parameters. The system is totally reactive and is able to satisfy a high level objective without the need for pre-planning. It can also operate in a “willed” mode under the direction of a high-level planner in environments where the dynamics of the world would not render it brittle (e.g. farms). Our use of a fuzzy arbitration mechanism, and the flexibility arising, is significant addition to earlier work such as the subsumption architecture [5].

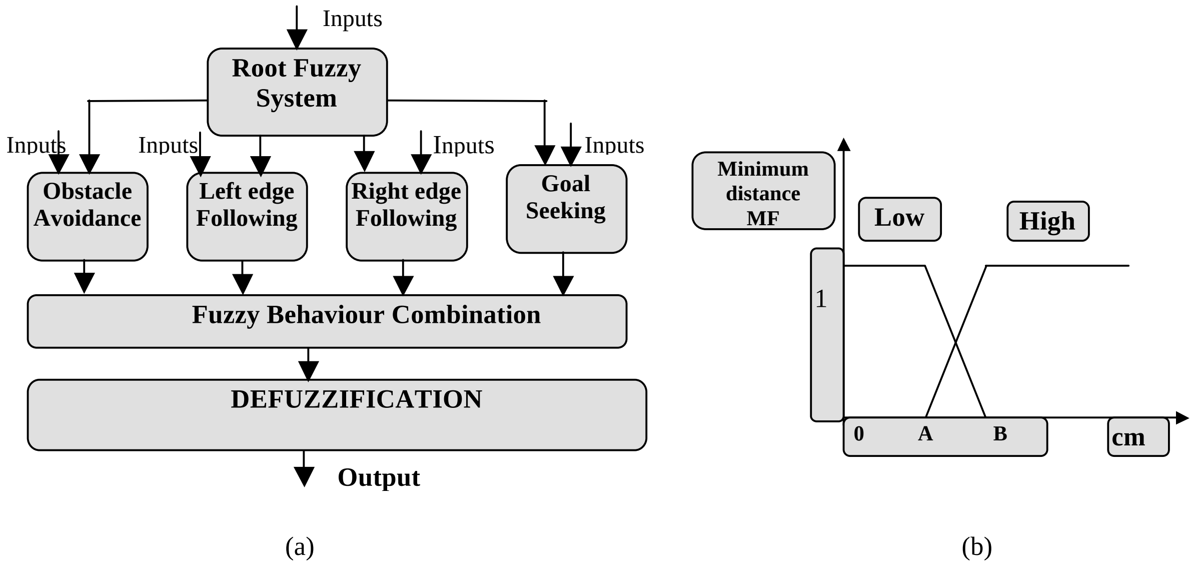


Figure (1): a) The behaviour co-ordination system. b) The membership function for the co-ordination parameters.

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 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:

$$Y_t = \frac{\sum_{p=1}^{M} y_p \prod_{i=1}^{G} \alpha_{Aip}}{\sum_{p=1}^{M} \prod_{i=1}^{G} \alpha_{Aip}} \qquad (1)$$

Where M is the total number of rules, y_p is the crisp output for each rule, $\Pi\alpha_{Aip}$ 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 [24,25].

Ruspini [33] defines fuzzy command fusion as the interpretation of each behaviour-producing unit, acting as an agent, expressing preferences as to which command to apply. Degrees of preferences are represented by a possibility distribution (fuzzy as in our case) over the command space. In our HFLC architecture a fuzzy operator is used to combine the preferences of different behaviours into a collective preference. Accordingly, command fusion is decomposed into two steps: preference combination and decision making. In Figure (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 [34] 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 case of using fuzzy numbers for preferences, product-sum combination and height defuzzifcation. The final output equation, according to Saffiotti [34], is given below:

$$Y_{ht} = \frac{\sum_i (mm_y * y_t)}{\sum_i mm_y} \qquad (2)$$

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 (left and right velocity in case of indoor robots, steering and wheel speeds in case of outdoor robots). 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.

The behaviours implemented in this system are the minimum set of behaviours we need to demonstrate this architecture working in an outdoor environment. From other work [5] we know that four behaviours are sufficient, namely 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 is 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 Figure (1-b). These fuzzy values are used as inputs to the context rules which are: *IF d1 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 Equation (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 Figure (1-b). Then, by using the context rules we can determine which behaviours are active and apply Equation (2) to obtain the final output.

3. Hierarchical Fuzzy Genetic Systems for Online Learning and Adaptation

To achieve a high-level behaviour via manipulating rule-bases with a GA, it is necessary first to 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 [10].

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 need to find next the most suitable membership functions for the learnt rules as was explained in [15]. After finding a sub-optimal solution for each behaviour we 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 was explained in [16]. After learning the system parameters, the controller then operates in its environment where the online adaptation technique is triggered to adjust the learnt controller to any environmental or kinematics changes or to any new situations it might encounter. If the controller fails to maintain the desired states, the online adaptation technique modifies the poor rules in the relevant behaviours to adjust to the agent differing environmental and kinematics conditions without the need to restart the learning cycle. This is called life long learning where the agent can adapt itself to any new situation and it 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 at 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. The life long learning is responsible for some degree of adaptation at the individual level. It works by tuning up 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 ones as life long learning presents the agent with very different perceptual stimuli than learning in a condensed period of time [30]. 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. Then we use our online adaptation technique to implement the life long learning where the agent is always updating its knowledge and gaining experience and is able to adapt to its 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 learn general controllers that can be applied to different robots performing the same mission and applying a short adaptation cycle thus saving the need to learn a new controller for each different outdoor robot.

Figure (2) provides an architectural overview of our techniques, which we term as *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 *Co-ordinator,* 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 behaviors, then the input membership functions of each behavior 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 at each step on small parts of the search space only. 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 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 will be done in our case by dealing with local models in the form of set of rules rather than dealing with big 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 [27].

To further reduce the search space, the system determines if it had encountered similar situations before by checking the stored experiences in the *Experience Bank.* The robot tests different solutions from the *Experience Bank* by transferring the "remembered" experiences, which are stored in a queue to the corresponding behaviors. If any of these experiences show success, then they are stored in 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 the new learning cycle. We then fire an *Adaptive Genetic Algorithm (AGA*) mechanism using adaptive crossover and mutation parameters which help to speed the search for new solutions. 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 and avoided searching regions where solutions are not likely to be found. After generating new solutions wither rules or MF or coordination parameters the system tests the new solution and gives it fitness through the *Solution Evaluator* to be discussed next section. The AGA generates new solution until reaching a satisfactory solution

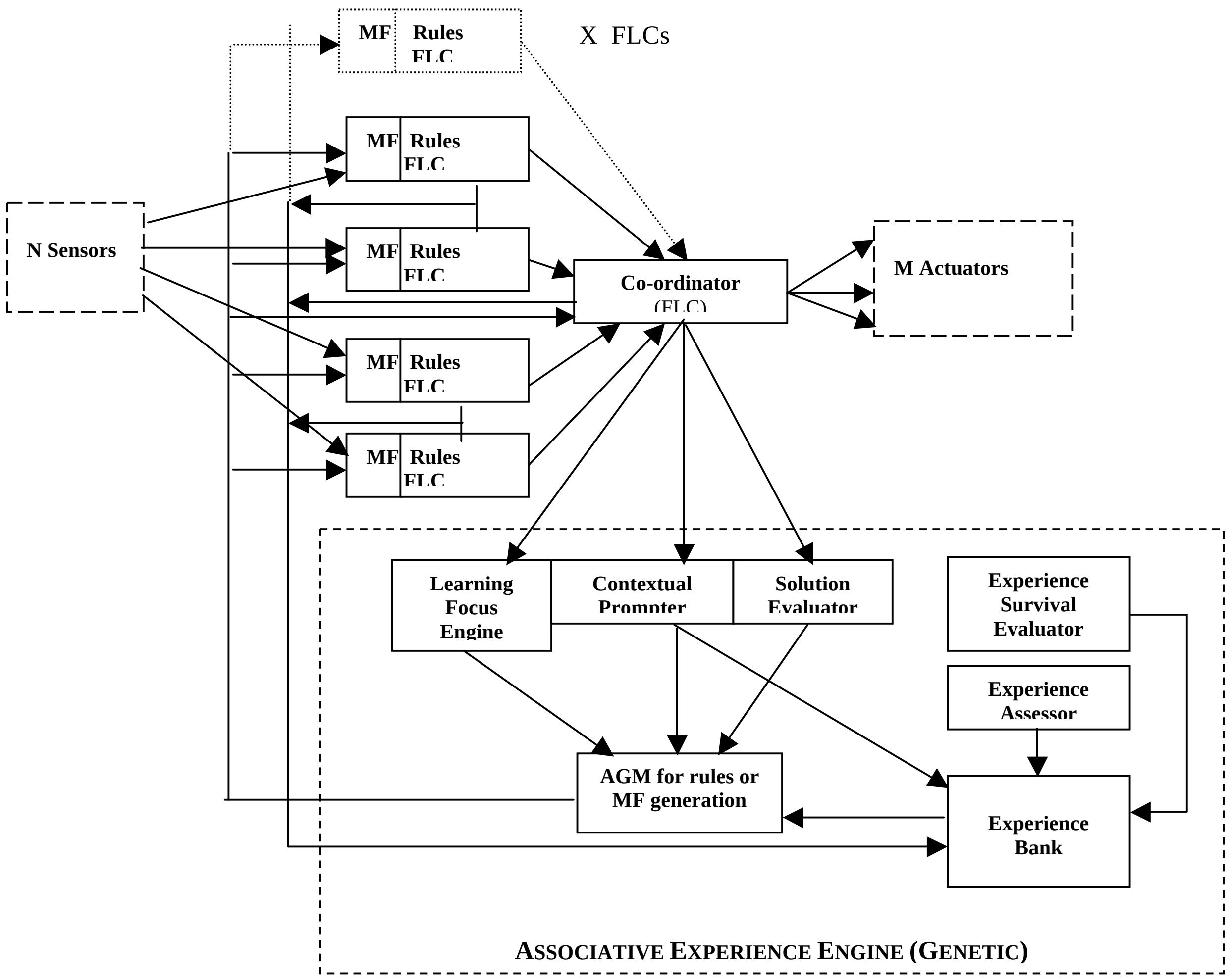


Figure (2): Architectural Overview of Associative Experience Learning Engine (British patent No 99-10539.7).

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 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 [11, 12, 13, 14]. For more information about learning the membership functions for each behaviour please refer to [15]. For more information about learning the co-ordination parameters online, please refer to [16]. Note that all the controller parameters are learnt online and through interaction with the environment with no need for simulation or human intervention in a short time interval.

3.1 Our Patented Techniques used to learn the obstacle avoidance behaviour

3.1.1 Identifying Poorly Performing Rules

After the rule-base initialization with random rules the robot starts moving using the bad rule-base, until the behaviour fails. In case of the obstacle avoidance behaviour this is when the robot is too close to an obstacle or a wall, as sensed by low range sonar or a bumper switch. As a result of a series of movement vectors {m(0), ….., m(t-3), m(t-2), m(t-1), m(t)}, the mobile robot may collide or get very close to an obstacle. To avoid such failures in case when the mobile robot navigates again through the environment similar to the previously experienced one, the rules contributed to generate the control action must be corrected. The rules responsible for the mobile robot failure, in turn will be the rules used in the previous time steps t, t-1, t-2,.. . Thus such rules must be changed into the correct rules [1].

The on-line learning algorithm is then fired to generate new set of rules to escape from this failure. As in classifier systems, in order to preserve the system performance the GA is allowed to replace a subset of the classifiers (the rules in this case). The worst m classifiers are replaced by m new classifiers 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 [8]. In our case, only 4 rules consequences will be replaced and these rules are the most effective rules in the situation of the behaviour failure (collision), because they are the mostly blamed for this failure. These rules are found by using the robot's Short Term Memory (STM) which is composed from the last 2000 actions (bounded by the memory size) the robot had taken before failure (colliding with an obstacle or a wall). When the robot begins moving using the new generated rule-base the STM is initialized to record the new 2000 actions. The robot replays the last 2000 actions from STM to begin moving backward along the original path and for finding the rules that caused the failure.

The distance backed at this step will be used as the starting point for all the solutions proposed by the algorithm. The method of specifying the rules to be replaced will be explained in detail later.

As mentioned before the robot starts its operation by moving using the initial rule-base until it is so close to an obstacle. At the moment of collision the robot begins backing off by replaying the actions that are stored in the STM, starting from the last action it had taken.

It is desired to determine the distance the robot backs off to avoid the behaviour failure again and to enable us to find the blamed rules for this failure. In the following analysis we will try to make all the computations related to the robot dimension, so that when moving from large robot to small robot, the algorithm can still work but with changing some parameters that depend on the robot dimensions. This means that our algorithm is *robot independent* and it is not developed for a specific kind of robot.

As getting so close to an obstacle requires steering away from it, it is required to find the minimum distance that if the maximum steering angle is applied the robot could pass without hitting the obstacle.

We define (W) to be the minimum safe front distance from an obstacle. At this distance if the robot applied the maximum steering angle it would avoid hitting the obstacle. The minimum value (W) would be the width of the robot, as shown in Figure (3). The robot also must satisfy that at this point the left and right side will also be safe at turning. This can be satisfied by making the minimum distance from the left walls (X1) equal to L1-W and from the right side X2 equal to L2-W, where L1 and L2 are lengths of the sides of the robot. The robot then backs off until the minimum reading of the front sensors is equal to or just greater than W, and the minimum reading of the left sensors is equal to or just greater than X1, and the minimum reading of the right sensors is equal to or just greater than X2. In this location if the robot applies the maximum steering angle, it should avoid the obstacle safely. We will call this distance the First Backing (FB). This technique is also efficient when encountering dead ends or when the space is tight for the robot to manoeuvre, in this case the robot will go back until it is possible for it to manoeuvre. At this location the robot must try maximum steering to get out of this situation, while if it backed more, it can apply less steering and get out of this situation. 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. So the robot will back another distance double the FB (using the values stored in the STM) and we will call this Second Backing (SB). At the end point of SB the robot will stop backing and consider this point its starting point of all the next iterations.

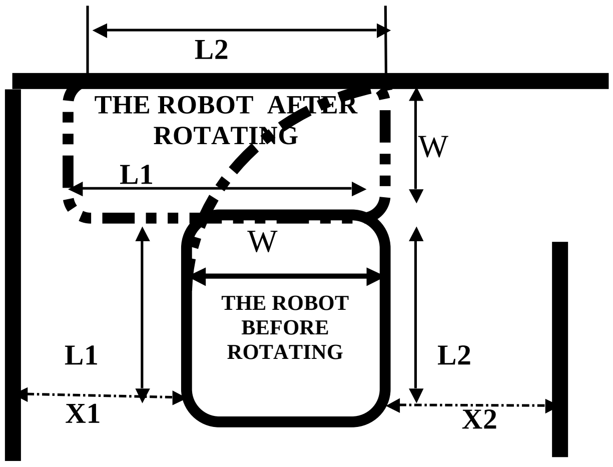


Figure (3): The robot turning distances.

As mentioned earlier, we cannot replace all the rules in the population, so we will replace only a part of the population. We will choose to replace the most two effective rules in each backing, these rules are blamed, because if they had taken the right actions, the robot can avoid the behaviour failure. So at FB the robot stops and finds all the rules that fired at this situation and evaluate the strength of each rule by how much it contributed to the final action. The greater the contribution the more it will be blamed for the failure and it will punished by reducing its initial fitness with respect to other rules, the most two effective rules (lowest fitness) consequents will be replaced later by two new rules consequents. The robot then backs and at SB it does the same for the rules at the SB situation, and the most two effective rules (lowest fitness) consequents will be replaced later by two new rules consequents. The population of GA is composed of all the rules that have contributed to the actions at FB, SB.

3.1.2 Fitness determination and Credit assignment:

The system fitness is supplied by the *Solution Evaluator* and is evaluated as the distance moved by the robot from its starting point before failure (collision). To determine this distance we use triangulation between 3 infrared beacons placed at known locations in indoor environments. This enables us to determine the approximate distance the robot had travelled before the collision occurred. This technique can also be applied to the outdoor robots using an electronic compass or a GPS sensor.

In this paper, we are concerned with making the robot learn online to 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 reflects 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, smooth transition between close models. Also dividing the learning into situations can reduce the number of learnt rules. For example if the learning is started by 25 rules, the robot might discover that during its training in its environment, it needed only 9 rules to navigate safely.

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 agent's performance when it succeeds in escaping, or when it is collides or gets close to an obstacle. We evaluate the performance of an agent after a sequence of control steps called *episodes*. This evaluation strategy averages the effects of the single rules, and in general it has a stabilising effect [2]. At the end of each episode, the reinforcement program evaluates the agent's performance and it distributes the corresponding reinforcement to the rules that have contributed to control actions at FB and SB.

In the following actions we will not use the Bucket Brigade algorithm for apportionment of credit assignment. As discussed in [20], the bucket-brigade algorithm may loose effectiveness as action sequences grow long, and as we are using the FLC system we have long chains of rules. Also the bucket brigade design doesn't appear to handle partially observed environments robustly [20]. So we will only apply credit assignment to the rules in FB and SB, as it will be shown that modifying these rules is sufficient to find a solution and there is no need for backward chaining.

We can write the crisp output Y_t as in (1). If the agent has N output variables, then we have Y_{t1} and $Y_{t2\ldots}$, Y_{tn} The normalised contribution of each rule p output (Y_{p1}, $Y_{p2}\ldots\ldots Y_{pn}$) to the total output $Y_{t1}, Y_{t2}\ldots, Y_{tn}$ can be denoted by S_{r1}, $S_{r2}\ldots.$ S_{rn} where S_{r1} , $S_{r2}\ldots.$, S_{rn} are given by:

$$\mathbf{S_{r1}} = \frac{\dfrac{Y_{p1}\prod_{i=1}^{G}\alpha_{Aip}}{\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{Aip}}}{Y_{t1}} \quad , \quad \mathbf{S_{r2}} = \frac{\dfrac{Y_{p2}\prod_{i=1}^{G}\alpha_{Aip}}{\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{Aip}}}{Y_{t2}} \quad \ldots\ldots., \mathbf{S_{rn}} \quad \frac{\dfrac{Y_{pn}\prod_{i=1}^{G}\alpha_{Aip}}{\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{Aip}}}{Y_{tn}} \tag{3}$$

In our robots we have only two outputs, which are left and right wheel velocities in the indoor robots and the speed and the steering in the outdoor robots therefore we have S_{r1}, Sr2.We then calculate each rule's contribution to the final action $S_c = \frac{S_{r1}+S_{r2}}{2}$. Then the most two effective rules in case of FB or SB are those rules with the greatest values of S_c. If there is an improvement in the distance, then the rules that contributed most must be given more fitness to boost their actions. If there is no improvement then reducing their fitness w.r.t to other rules punishes the rules that contributed most 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:

$$\mathbf{S_{rt}} = \text{Constant} + (d_{new}-d_{old}) \,.\, \frac{S_{r1}+S_{r2}}{2} \,.\, (1-F).\, V \tag{4}$$

where d_{new} is the distance moved by the robot after producing a new rule-base by the online algorithm, d_{old} is the distance moved by the robot from the previous iteration, d_{new}-d_{old} is the distance improvement or degradation caused by the adjusted rule-base produced by the algorithm. F is the normalised robot steering. V is the normalised average speed of the two wheels. This makes S_{rt} maximised by increasing the distance moved by the robot and higher speed with minimum adjustments to steering. The variable V was introduced to favour rules with faster speeds and F was introduced to penalise instant large steering variations and zigzag paths, which means that better fitness is obtained when the robot tries to move forward with minimum deviation not moving in zigzag paths.

In the first population of GA, as there is no distance moved yet, we blame only the rules that have contributed more for the action of collision and the fitness of each rule is given by:

$$\mathbf{S_{rt}} = \text{Constant} - \frac{S_{r1} + S_{r2}}{2} \qquad (5)$$

In this way the rules that have contributed more to this bad action will have lower fitness value than the rules that have contributed less to this action which allows the GA to go away from these bad actions and begins exploring other actions.

3.1.3 Experience Bank Application

Zhou [45] presented the CSM (Classifier System with Memory) system that addresses the problem of long versus short-term memory, i.e. how to use past experiences to ease 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. The short term memory is just the standard set of rules found in every learning classifier system; the long term memory is a set of rule clusters, where every rule cluster represents a generalised version of the problem solving expertise acquired in previous problem solving activity. Each time the agent is presented with a problem it starts the learning procedures trying to use long term experience by means of an appropriate initialisation mechanism. Thereafter, the system works as a standard classifier system (except for some minor changes) until an acceptable level of performance has been achieved. It is at this point that a generalizer process takes control and compresses the acquired knowledge into a cluster of rules that are memorised for later use in the long term memory.

In our system, as the robot begins the motion, it has no previous experience and the *Experience Bank* is empty. But as it begins learning by GA, it begins filling the memory with clusters of rules. Each rule cluster is consisting of the rules that were learnt and the actions (consequences) that were learnt by the GA. The rules are stored in a queue based on recent experience.

Each time the robot is presented with a situation to solve, it begins checking if the rules to be modified are present in the memory clusters or not. 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 2,4 will remain the same. Then the robot begins moving with this modified rule-base. If it survives and gets out of this situation with no collisions or failures, then these 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 had worked in different environmental conditions. If the robot fails (collides with an obstacle) again, it measures the distance it had moved to determine the fitness of the solution proposed by this memory cluster (supplied from the *Experience Bank*). After all the memory clusters have been examined and the robot still collides. The best solution proposed by the *Experience Bank* according to its fitness based on how long it had moved before colliding is kept in the rule-base of the controller, in order to serve as a starting position of the GA search instead of starting from a random point. The Experience Bank actions which act as a long term memory will serve to speed up the search.

By doing this, our system does not need the matcher calculations used by [45]. This is because as our system does not use the binary message coding or "don't care" conditions rather using perfect matches, hence we don't need the generalizer. The clusters are laid in a queue starting from the recent experience.

The problem occurs as the system begins accumulating experience that exceeds the physical memory limits. This implies that we must get rid of some of the stored information as the acquired experience increases. However we don't favour this, as this means that some of the experiences acquired by the robot will be lost. So for every rule cluster we attach a *difficulty counter* to count the number of iterations taken by the robot to find a solution to a given situation, we also attach a *frequency counter* to count how much this rule have 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. This is similar to a situation of a sinking boat where we have to sacrifice some of the passengers and keep others according to their relative importance.

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 initialised 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 robots), so if we exceed this limit the *Experience Survival Valuer* begins using the *degree of importance* and the age operator to optimise the Experience Bank. Of course not all the 2000 clusters will be used in each trials, because we will use the clusters that contains the rules that match the current situation.

3.1.4 Producing New Solutions by GA

After the failure of the actions acquired from the Experience Bank, the GA starts its search for new consequences for the blamed rules. The fitness of each rule in the population is proportional to its contribution in the final action. If the proposed action by the new solution results in an improvement in the distance then the rules that have contributed most will have their fitness increased more, than the rules that have contributed less in this situation. If the result was a decrease in the distance moved before collision then the rules that have contributed most to this action will have their fitness less than the rules that have contributed less to this action. This allows us to go away from those points in the search space that cause no improvement or degradation in the performance.

The parents for 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. The crossover is selected to be 1.0 by empirical experiments and the mutation is variable according to the improvement in the distance. If there is no improvement or there is degradation then the mutation is set to high probability chosen to be 0.5 by empirical experiments, which means that we want to introduce new genetic materials in the solution. If the result was an improvement then the mutation rate is lowered proportional to this improvement until the fitness reaches a high limit (to be discussed later). If the mutation is set to zero it means we want to keep this genetic material with no disruption and to fine tune the solution using crossover. Binary coding is used in coding of the chromosomes.

After the *Adaptive Genetic Algorithms* (AGA) generates a new solution, the robot tries the controller with the modified rule-base, if the robot had moved a certain distance without crashing (to be determined later), then this is an ending criteria. This means that the robot had learnt this situation. The robot keeps this solution in the rule-base and in the Experience Bank. If not it goes back to section 3.1.2, skipping section 3.1.3. In the following section we will explain these steps in more detail.

The proposed system can be viewed as a double hierarchy system in which the fuzzy behaviours are organised in a hierarchical form and the online learning algorithm is also a hierarchy. In the higher level we have a population of solutions stored in the *Experience Bank* and they are 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 assigned by the *Experience Assessor* by finding the distance it had moved before colliding. The highest fitness experience is used as a starting position to the lower level GA that is used to produce new solution to the current situation.

The 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 the solutions stored in the Experience Bank fail. The AGA produces new solutions that replace the most two dominant rules at FB and the most two dominant rules at SB.

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 all the rules actions of FB and SB. This means that modifying the SB rules are not enough to solve this situation and it is needed to take into consideration the FB rules. We will call this Second Replacement (SR).

The robot then starts moving with the modified rules. If the robot moves a distance more than the distance needed for the ending criteria of this situation (to be determined later), this will be considered a solution, until it collides again (requiring it to learn a new situation). If the number of generation exceeds a maximum number of generations chosen empirically to be six with no solution 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 Figure (4). Splitting this situation into two sub-situations is essential for producing a solution.

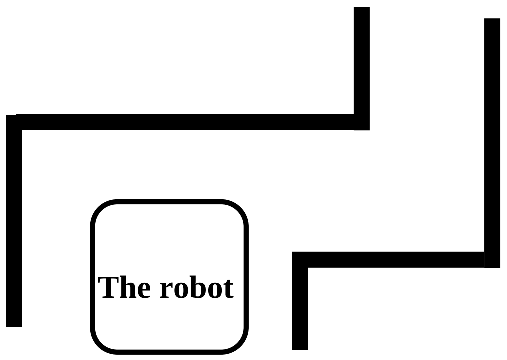


Figure (4): The robot is in a situation composed of two sub-situations, one is right turn followed by left turn.

The population of the GA during the FR will be the actions of all the rules that have contributed to the SB (which is usually a small population of 6-12 rules depending on the situation). While in the SR the population will be consisting of all the rules that contributed to the FB and the SB.

The crossover and mutation probabilities play a major role in the fast convergence of the GA. The selection procedure for these probabilities will be discussed in detail. The crossover probability P_c controls the rate at which the solutions are subjected to crossover. The higher the values 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. Large values of P_m transform the GA into a purely random search algorithm, while some mutation is required to prevent the premature convergence of the GA to sub optimal solutions. 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 [37].

As we are using a small population size and small chromosome size, we need a high mutation rate to allow for a wider variation in the search and hence the ability to jump out of any local minima. Therefore we need a high mutation rate so as to introduce new genetic material without reducing the search process to a random process. It is also desirable as the system shows improvement in fitness (distance), that mutation rate is decreased for not losing the genetic materials that caused the improvement and we depend on crossover to fine tune these genetic materials to obtain a the desired solution.

So the mutation probability we propose will be variable from one generation to the other, and it will depend on the distance improvement. In the first generations we will use a high mutation probability found empirically to be 0.5 (to be shown later). If there is an improvement in the distance, we will linearly reduce the mutation until the improvement reaches a maximum value. If the distance improvement was the same or was less than the previous trial, then the mutation rate is increased again to 0.5 to find new genetic materials that might aid in finding a solution.

So the mutation probability will be given by:

$$p_m = 0.5 - \frac{0.5(d_{new} - d_{old})}{robotlength} \qquad \text{if } \boldsymbol{d}_{\text{new}} > \boldsymbol{d}_{\text{old}}$$

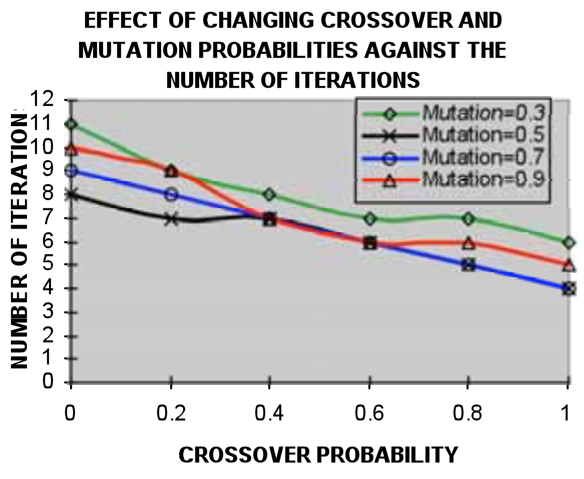
$$p_m = 0.5 \qquad \text{otherwise} \qquad (6)$$

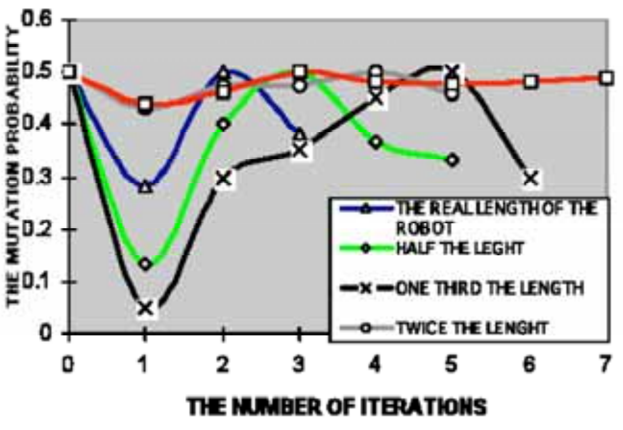
The reduced crossover lowers the productivity of the GA, since there is less recombination between individuals, and hence it takes a longer time to obtain good solutions, so as in [27] we will set the crossover probability to 1.0 to guarantee fast convergence. The above selections will be justified by the following experiments.

In the following experiments we wanted to find a solution for the problem encountered by the robot in which it is required to learn only the right turn in a corridor. In Figure (5-a) we have conducted different experiments for each mutation value we have tried 6 values of crossover probability starting from 0 to 1 with a step of 0.2. And then we varied the mutation probability starting from 0 to 1 with values equal to 0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0. It was found that at zero mutation no solution could be found because lack of genetic material, the same was true for a value of 0.1. At a mutation value of 0.3, the fastest convergence was after 7 iterations with crossover value of 1.0. At a mutation value of 0.5, we have the fastest convergence after 4 iterations with a crossover rate of 1.0. The same convergence rate was noted for mutation values of 0.7. At mutation values 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 mutation rate of 1.0 leads to no solutions at all. This is because starting with most the genetic material the same, mutation will lead to inversion of the binary material and we will end with all the genetic material the same and random search and no new material will be introduced.

From this Figure (5-a) it is obvious that always as the crossover rate increases, the convergence rate is faster with an optimum value achieved at a crossover probability of 1.0. Also the optimum mutation value was found to be either 0.5 or 0.7, but we will choose 0.5 to be the bound to decrease the risk of ending in a randomised search.

We conducted a series of experiments to investigate the effect of the robot length variation in Equation (6). This would determine if the robot length was an optimum parameter? That is to say if we tricked the robot by saying that its size is half its original size or double what will be the effect? By doing this we can investigate the value for the optimum parameter in Equation (6). From Figure (5-b), it is obvious that the original robot size had given the fastest convergence (after 4 iterations) whilst any other lengths didn't give the same fast convergence (while maintaining the crossover =1 and mutation using Equation (6)).





(a) (b)

Figure (5): a)The convergence rate specified by the number of iterations plotted against the crossover probability for different mutation rates. b) The effect of variation of the robot length in Equation (6) over the rate of convergence (the number of iterations).

In order to be sure that this parameter is optimum we tried varying the robot sizes and mazes size. By doing this we were investigating that the effect of the robot length parameter was not a parameter dependent on the maze and that by changing the maze size, the optimum parameter was always the robot original length.

We use binary coding in the AGA. For each rule there are two actions (which are the left and right wheel velocities in case of the indoor robots and velocity and steering in case of the outdoor robots). As we have 4 output membership function, we decode each action by two bits as follows, *Very Low* is 00, *Low* is 01, *Medium* is 10, *High* is 11. By doing this we have a chromosome length of 4 bits.

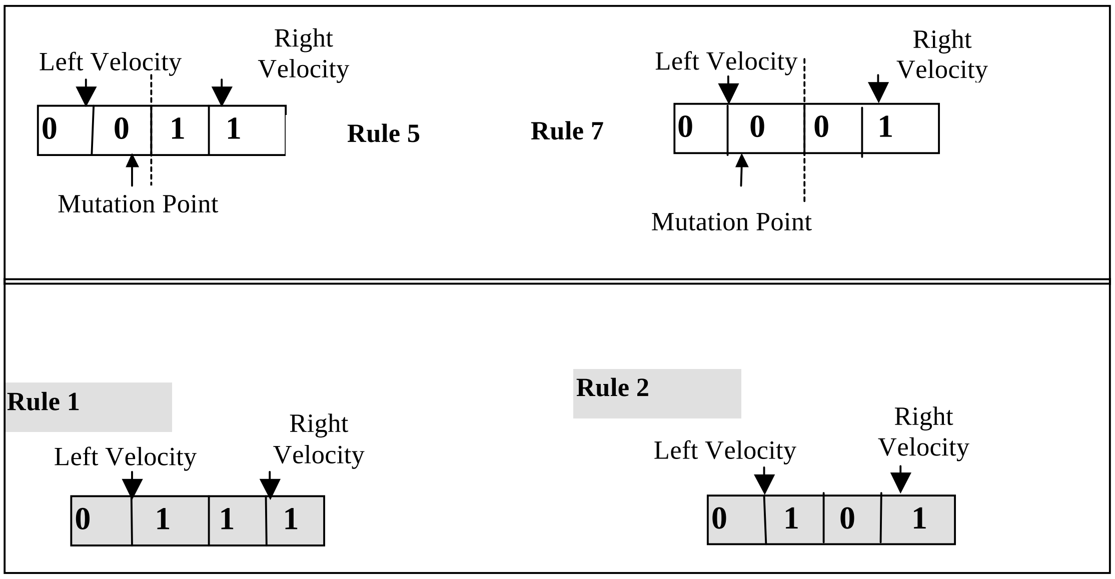


Figure (6): An example of GA processes in our system in which rule 5 and rule 7 (which were selected due to their higher fitness values) are generating new consequent for rules 1,2 of the SB reversal. The same will happen with two FB rules.

Figure (6) shows a description of the AGA operations 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 contributed more with their actions to the final action which caused improvement, or contributed less with their actions to final action which caused degradation. The crossover of probability 1.0 was applied to both chromosomes and the adaptive mutation as well. The resultant off springs were used to replace the consequents of rules 1 and rule 2 which were mostly blamed at SB. The same technique is used to replace the consequents of the two dominant rules in the FB.

However to speed up the search we will use the *Contextual prompter Constraints*. This uses the sensors information in order to narrow the search space of the GA and 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 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 time when using *Contextual Constraints* as the system is searching through the whole search space not in regions where the solutions is likely to be found.

3.1.5 Ending Criteria

At the SB the robot is at minimum 2W from the front obstacle and X1 from the left and X2 from the right. If it is assumed that the robot moved W without doing the correct action and at the position of FB it made the correct turn, this should be rotating with quarter a circumference of a circle of radius W making the robot moving $\frac{\pi . W}{2}$. In order to make sure it is out of this situation with no collision, the robot should escape with its sides L (maximum of L1, L2). Then the total distance moved by the robot to get out safe of a situation without hitting an obstacle is given by:

$$W + \frac{\pi . W}{2} + L \qquad (7)$$

So the robot calculates the average distance moved, if this distance exceeds the distance given by Equation (7), then the robot had successfully found a solution to this situation. If the number of generations exceeds 6 then the distance given by Equation (7) is reduced by half, to split this situation into two situations as described earlier.

3.2 Online Adaptation of the learnt controller and Life Long Learning

In the previous sections we explained how to learn the rule base for the obstacle avoidance behaviour online and through interaction with the environment using the Evolutionary techniques, imitating the natural evolution and using our patented techniques to speed it up so that we can learn online. In this section we discuss how can we adapt the robot to any environmental changes and implementing a life long learning scheme where the agent gains and updates its knowledge by encountering new situations. This means that the robot can add or delete or modify rules according to the situations it encounters.

Also this approach is useful in prototyping as we can learn the robot controller on a prototype robot in a controlled environment and then we transfer it to the agent running in the real environment where we only run a short adaptation cycle with no need to repeat the learning cycle. Our algorithm parameters were robot independent, so that the algorithm can be moved between the prototype and the real agent with only minimal changes. After transferring the prototype controller to the target, the target then begins running an online adaptation module that allows it to adapt to the differences between the prototype and the target robots and environments. This procedure is useful in learning controllers for robots 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 embedded robots. Also our learning techniques learn general controllers that can be applied to different robots performing the same mission and applying a short adaptation cycle thus saving the need to learn a new controller for each different robot. Perhaps more importantly, this adaptation session is important in the case that environmental or robot kinematics changes occur and the robot needs to rapidly adapt to these changed circumstances.

Although fuzzy logic allows a degree of imprecision to exist within the environment, significant environmental or agent 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 is difficult if not impossible for dynamic outdoor environments). An alternative solution is to allow the existing co-ordinated rule set to be adapted to compensate for the environmental differences. This latter approach only requires a basic rule set to be present from the prototype robot controller and does not require prior knowledge of all the possible environments. We start the adaptation from the best learnt HFLC by the robot, instead of starting from a random point. If the system was started with random rule bases, the system could still modify its actions but would take approximately six times the time needed when started from a good rule base containing some inappropriate rules for the new environment. This figure was found by practical experimentation.

The system discovers the rules (in different behaviours) that, if modified, can lead the robot to adapt to its environment. Whilst it might seem a good idea to change the co-ordination 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, then clearly changing the co-ordination parameters will never correct the robot behaviour (as was proved by experimentation). Also changing the MF is a difficult task, as it needs to be done to each individual behaviour necessitating the robot to be taken away from its environment for MF calibration. However, our method allows the poor rules to be found and corrected for each behaviour, online and in-situ without the need to repeat the whole learning cycle, modifying only the actions of a small number of rules that performed poorly.

Imagine a situation where the robot had failed to do its mission. In this case we will modify all the rule bases whose behaviours were active. The robot then uses its short term memory to return back to its pre-failure position, identifying the two most dominant rules (which can belong to different behaviours). The robot then replaces the actions of these rules to solve this situation. The way the robot determines the dominant rules is done by calculating the contribution of each rule to the final output in a given situation as follows: Apply Equation (2) and substitute Y_t from Equation (1). Then we can write the crisp output Y_{ht} as:

$$Y_{ht} = \frac{\sum_{y=1}^{4} mm_y \dfrac{\sum_{p=1}^{M} y_p \prod_{i=1}^{G} \alpha_{Aip}}{\sum_{p=1}^{M} \prod_{i=1}^{G} \alpha_{Aip}}}{\sum_{y=1}^{4} mm_y} \qquad (8)$$

Where M is the total number of rules, Y_p is the crisp output for each rule, $\Pi\alpha_{Aip}$ 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.

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

$$\mathbf{S_{ra11}} = \frac{\dfrac{mm_y}{\sum_{y=1}^{4} mm_y} \cdot \dfrac{Y_{p1}\prod_{i=1}^{G}\alpha_{Aip}}{\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{Aip}}}{Y_{ht1}}, \quad \mathbf{S_{ra22}} = \frac{\dfrac{mm_y}{\sum_{y=1}^{4} mm_y} \cdot \dfrac{Y_{p2}\prod_{i=1}^{G}\alpha_{Aip}}{\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{Aip}}}{Y_{ht2}}, \quad \mathbf{S_{rann}} = \frac{\dfrac{mm_y}{\sum_{y=1}^{4} mm_y} \cdot \dfrac{Y_{pn}\prod_{i=1}^{G}\alpha_{Aip}}{\sum_{p=1}^{M}\prod_{i=1}^{G}\alpha_{Aip}}}{Y_{htn}} \qquad (9)$$

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

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

In our robots we have only two output variables, therefore we have $Sa_{c1} = \frac{S_{ra11} + S_{ra22}}{2}$. The most effective rules, are those rules with the greatest values of Sa_{c1}.

In order to implement Life-Long learning the robot has a long time Experience Bank composed of all the experiences it encountered in the past and how the robot had succeeded to solve the problems it encountered. These experiences are in the shape of rule clusters having the rule number and the consequents the robot had learnt to associate with inputs of these rules in a situation where the agent failed to satisfy the desired behaviour. The rule clusters are arranged in a queue starting from the most recent experiences. These clusters represents how the agent have adapted to 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 its different actions. So in case of the agent failing to achieve the desired behaviour in the changing unstructured environment then the actions of the most effective rules can be modified to solve this situation. The agent then matches the effective rules to 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, AND the first rule cluster in the Bank contains the consequences of rules 1 of obstacle avoidance; rule 3 of left edge following and rule 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 agent starts operating with this modified rule-base taken from the *Experience Bank*. If it survives and gets out of this situation with no behaviour failures, then these 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 that had worked in different environmental conditions. This is the same as nature where the agent tries to recall its experiences that solved the same situation before. If none of these experiences had solved this situation, the robot assigns to each experience a fitness value indicating how well it performed in the current situation. Then the robot chooses the consequents of the rule clusters that have given the highest fitness and keep them in the rule-bases of the controller in order to serve as a starting position of the Adaptive Genetic Algorithms (AGA) search instead of starting from a random point, these consequents will be the parents of the new rule consequents. This action helps to speed up the genetic search as it starts the search from the best found point in the search space instead of starting from a random point.

The robot then calculates the fitness value of the solution starting from the modified rule base supplied by the *Experience Bank*. If there is an improvement in the performance 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 must be punished by reducing their fitness w.r.t to other rules 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:

$$\mathbf{S_{rat\,1}} = \text{Constant} + (d_{new}-d_{old}) \,.S_{ac1}.\,(1-F) \qquad (10)$$

Where d_{new} is the new performance of the robot using the modified rule base, d_{old} is the old performance of the robot before modifying the rule base, d_{new}-d_{old} is the performance improvement or degradation caused by the adjusted rule-base produced by the algorithm. F is the normalised robot steering. V is the normalised average speed of the two wheels. This makes S_{rt} maximised by improving the robot performance with high speed and minimum adjustments to steering. The variable V was introduced to favour rules with faster speeds and F was introduced to penalise instant large steering variations and zigzag paths, which means that better fitness is obtained when the robot tries to move forward with minimum deviation not moving in 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:

$$\mathbf{S_{rat}} = \text{Constant} - S_{ac1} \qquad (11)$$

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).

The AGA is the same as Section (3.1.4) if the behaviours causing failure contain the obstacle avoidance behaviour. 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 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 behaviour failure. The AGA is constrained to produce offspring according to the *Contextual Constraints* supplied by the input sensors in accordance with Section (3.1.4). The robot ends the adaptation when the agent achieves the desired response.

4. Experiments and Results

In the obstacle avoidance behaviour of both the indoor and outdoor robots each robot has three sensors which are the Left Front Sensor (LFS), the Medium Front Sensor (MFS) and the Right Front Sensor (RFS). The designer supplied a preliminary input Membership Function (MF) for each sensor composed of three fuzzy sets representing the Near, Medium and Far MF. The optimum value for MF was learnt in [Hagras 2000a]. The robot learns the obstacle avoidance rule-base by learning different situations (local solutions) while it navigates as was explained earlier in Section (3) and then the interaction among local models, due to the intersection of neighbouring fuzzy sets causes that local learning reflects on global performance. The robot does not learn special situations, but it learns general rules like if the *right sensor is low and the medium sensor is low and the left sensor is high then go left.* By encountering different situations the robot can fill its rule-base. So the training environment should be as complex as possible to supply the robot with many different situations to learn. The robot learns when it is needed. For example, if the robot was launched in a corridor, it will learn the rules needed to navigate in this corridor and it can generalise as will be shown later and navigate in different shapes of corridors, because it had learnt general and not specific rules. However, when the robot is introduced to a complicated maze with left and right turns the robot must learn more rules in order to survive. This also means that if after learning a complete rule-base, the robot changes its kinematics or the ground conditions change, the robot can still adapt itself to the environment by only adjusting small set of rules using our online adaptation and life long learning techniques explained in Section (3.2) without the need to start learning from the beginning as in learning by simulation.

The problem of the obstacle avoidance behaviour is that it cannot be employed as an independent behaviour (asking the robot to wander randomly and not collide). The obstacle avoidance behaviour can be regarded as a safety feature 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 we will co-ordinate the goal seeking behaviour whose rule-base was obtained from our previous work [11, 12, 13, 14]. For example, if the robot is asked to get out of a complicated geometrical structure and reach a target. In these cases where there is an obstacle in front of the robot which can be avoided from either the left or right, the obstacle avoidance rules can be produced that favour reaching the goal using the shortest path. In contrast to going in arbitrary directions, as suggested by other researchers [28], [44].

In our learning experiments we assume that the behaviour co-ordination membership function are the same as Section (2). The co-ordination context rule will be only reduced to: *IF d1 IS LOW THEN OBSTACLE AVOIDANCE, IF d4 IS HIGH THEN GOAL SEEKING.*

We have initialised the obstacle avoidance rule-base randomly to move the robot forward (biased to right or left, or with no bias with different speeds not including zero), this action was done in order to be sure that the robot is moving and did not remain stationary.

The robot learning cycle discussed above is shown in Figure (7-a). In this example the robot moves and fails, then it first backs (FB) and second backs (SB), and finally generates a modified set of rules for the situation. It can then pass safely until the rules fail again and the learning cycle is repeated.

The robot was first introduced to the complicated geometrical structure in Figure (7-b) which contains many general situations to learn, such as how to navigate in a corridor, how to do left turn and how to do right turn, and how to navigate in wide areas with dead ends.

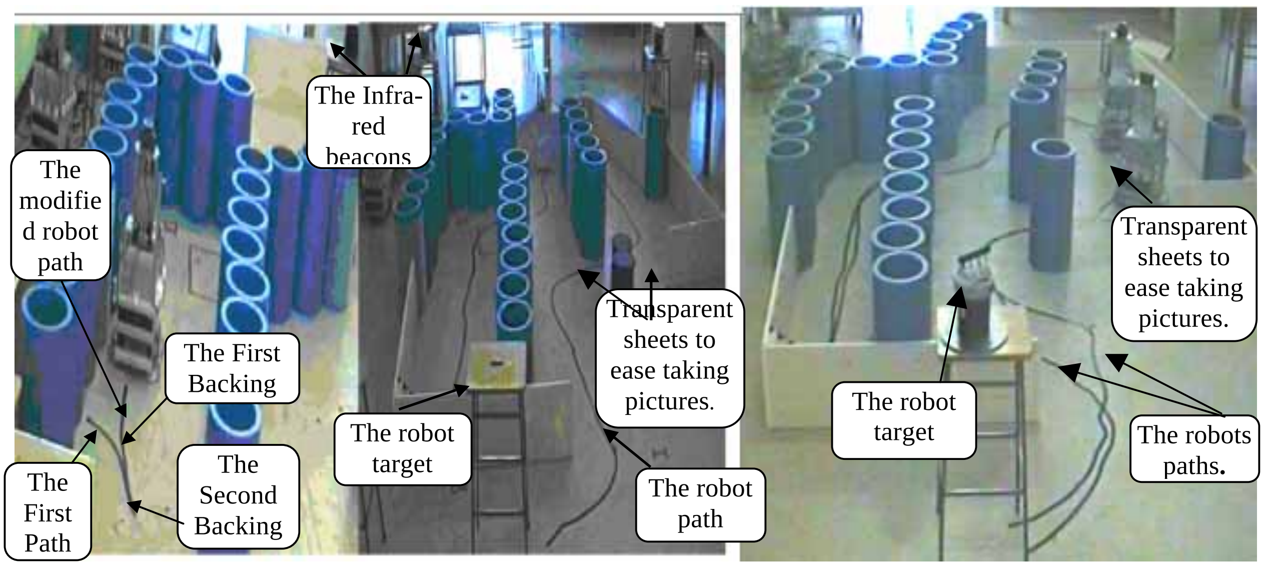


Figure (7): (a) The robot learning cycle. (b) The robot path after learning. c) Two different robots paths with two learnt rule-base

After only an average of 44 generations over 10 experiments (as it was found that the average is almost the same for values greater than 10 experiments) starting from random positions and with different initial rule-bases, the robot succeeded in getting out safely. The number of actions modified during the learning process was in average 15 rules in the obstacle avoidance behaviour. This shows that the algorithm had optimized the rule-base so instead of having to fill $3^3 = 27$ rules, the robot had found that it needed only 15 rules to complete the mission that resulted in rule reduction.

Although the experiment lasts for about 25 minutes, most of the time elapsed concerns moving backward and forward as this takes a long time due to the low speed of the robot. The computation time for each rule generation is 100 ms using a 68020 20MHz microprocessor. We have tried the robot with no *Experience Bank* and we 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, not from a random point.

Table (1) shows the best learnt rule-base which produced the faster time of getting out of the maze and with least normalised steering and minimum absolute average deviation from the desired safe distance. The robot response for this best learnt rule-base is shown in Figure (7-b).

Rule No	LFS	MFS	RFS	LEFT VELOCITY	RIGHT VELOCITY
1	NEAR	NEAR	MED	HIGH	VERY LOW
2	NEAR	NEAR	FAR	HIGH	VERY LOW
3	NEAR	MED	MED	HIGH	LOW
4	NEAR	MED	HIGH	HIGH	VERY LOW
5	NEAR	HIGH	MED	HIGH	VERY LOW
6	NEAR	HIGH	HIGH	HIGH	LOW
7	MED	NEAR	NEAR	VERY LOW	HIGH
8	MED	NEAR	MED	HIGH	VERY LOW
9	MED	MED	LOW	VERY LOW	HIGH
10	MED	HIGH	LOW	VERY LOW	HIGH
11	MED	HIGH	MED	VERY LOW	HIGH
12	HIGH	NEAR	NEAR	VERY LOW	HIGH
13	HIGH	NEAR	MED	VERY LOW	HIGH
14	HIGH	MED	NEAR	VERY LOW	HIGH

Table (1): The best learnt rule base of the obstacle avoidance behaviour.

After the robot gets safely out of the maze, we placed it in different starting position to test the repeatability and stability for 10 experiments. The robot has shown that the path is repeatable, following the same path produced during learning with an average deviation of 1.8 cm. 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. 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 had showed that the two solutions are statistically similar which shows that our system always can find a good solution and the difference between the best and the worst rule-base is small. For example, the best rule-base produced an absolute average deviation of the desired safe distance of 5 cm. While the worst rule 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. Also the learnt rule-base produced faster speed and less steering deviation than the manually designed one. For example in case of Figure (7-b) the learnt rule-base escaped the structure in average time of 75 seconds, while the manually designed rule-base escaped the structure in 104 seconds.

The robot's path shown in Figure (7-b) is a smooth path through the whole maze, getting safely out towards its target. We have also used two different robots (different robots kinematics and sensors positions) to learn the robots obstacle avoidance rule-bases. Each robot had produced 15 rules in average over 10 experiments started from different positions and with different initial rule-bases. Then the best performing rule-base was chosen. The robots response after learning is plotted in Figure (7-c), and they are almost the same. It is noted that each robot has a slightly different rule-base from the other as each robot has different characteristics from the other, and the robot tries to adapt its rule-base to its characteristics and its environment. This demonstrates that the learnt responses are related to the robot, and different robots can adapt themselves to their environments, this is why it is better to learn online rather than learning in simulation where it is assumed that all the robots have the same characteristics.

In other work based on simulation, the problem 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 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.

Figure (8-a) introduces the problem tackled by [26], [2] which is the conventional corridor tracking problem, and achieving a goal at the end of the corridor. We will compare the results first with [2] in which he placed a 60 cm wide robot in a 3m wide corridor before moving it to 4m and 2m wide corridors, then to the complicated corridor shown in Figure (8-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 m corridors and 83 cm 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, we also tried loosening the left wheel to simulate changing robot conditions. We started the learning by the corridor shown in Figure (8-c). It took the robot 16 minutes (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, and after learning it produced the path shown in Figure (8-a). The modified rules are shown in Table (2). 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 Figure (8-a) and Figure (8-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 [26] only experimented with simple corridor-following in simulation using context dependent coding succeeding in generating a solution after 40 generations (his rule-base would need to be modified to solve the problem of Figure (8-c)). Bonarini used his algorithm with a simulated robot before transferring his controller to a real robot. To solve the problem in Figure (8-d) he needed 471 leaning episodes (iterations) while we needed only 20.

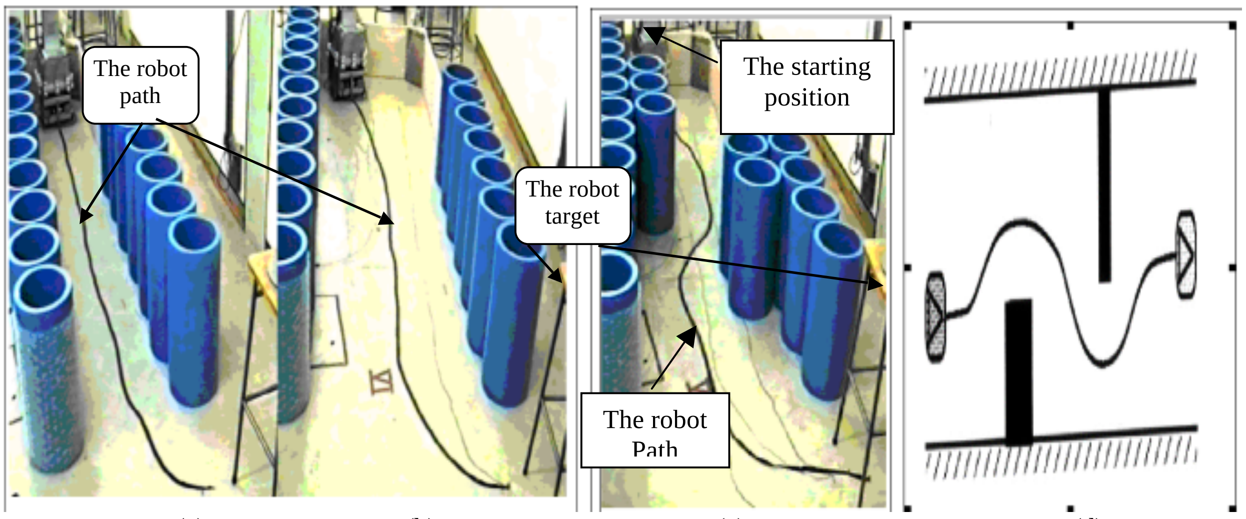


Figure (8): Leitch and Bonarini corridor experiments a) Tight corridor. b) Wide corridor. c) Our algorithm. b) Bonarini's method.

Hoffmann [19] introduces his method of incrementally tuning fuzzy controllers by means of an evolutionary strategy. In the benchmark problem of Figure (9-a), he had 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 we have destroyed the rule base causing all their actions to go left. We have fired the algorithms for modifying the obstacle avoidance rule base for Figure (9-a). After 18 minutes (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. The robot was subjected to 8 further trials to test its repeatability and stability. It followed the given a path with high repeatability and stability. The robot is reactive as shown in Figure (9-b), (9-c), as the target changes its position from the left to the right, the robot changes its path responsively following the shortest path.

	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	High	Near	Low	Med
	Far	Far	Near	Very Low	High
	High	Med	Near	Low	High
	High	High	Near	Very Low	Med

Table (2): The modified rules for the experiment in Figure (72-a).

The same controller from the previous experiment was applied to the problem in Figure (10-a). In this 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 Figure (10-b). This supports the generality of the learnt rules.

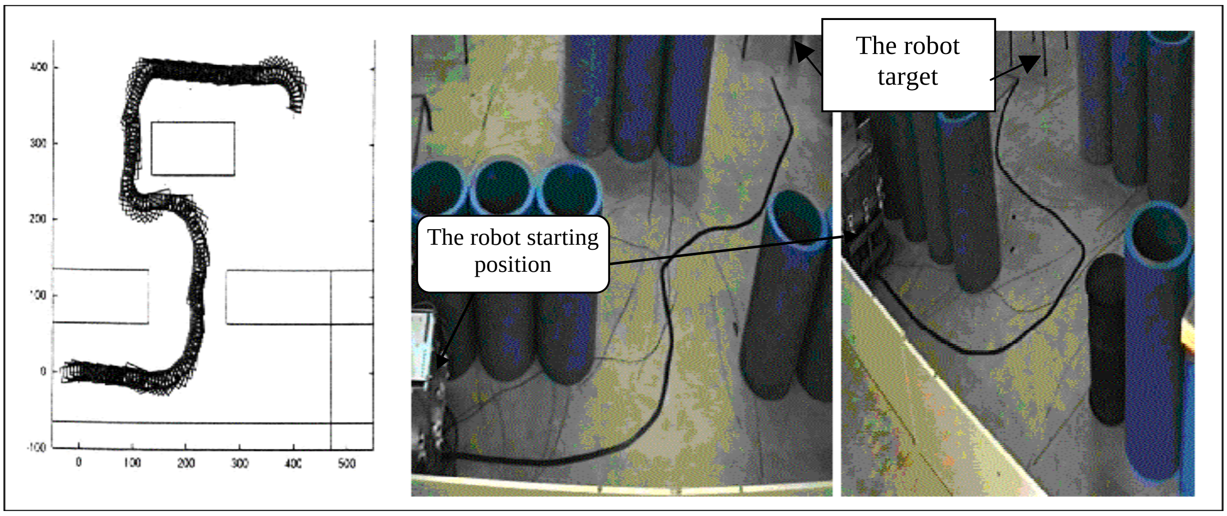


Figure (9): Comparison to Hoffmann's work a) Hoffmann's method. b) My method with target to the right. c) My method with the target to the left.

Figure (10-c) shows the robot with the behaviours learnt from the previous experiment and applied to different maze structure. Again the robot displays a good response escaping the maze and achieving its goal.

The rule bases generated starting from different starting positions and different rule bases are slightly different, they both modified almost the same rules and gave a comparable response. It is difficult to say which one is better from the other as statistically, according to the t-test, both solutions are similar. This indicates that while the robot modifies its obstacle avoidance rule base online, any learnt rule base will have the same effect as other learnt rule-bases, supporting the generality of our solution.

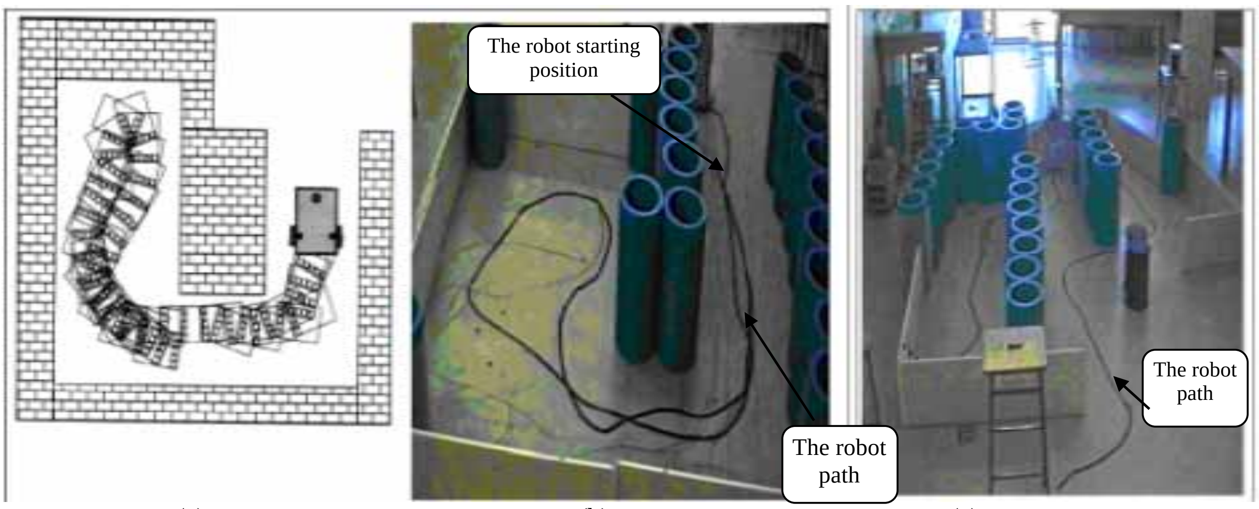


Figure (10): Comparison to Hoffmann's work a) Hoffmann's method. b) Our method. c) The robot response starting from a different position.

After learning the rule base for the obstacle avoidance behaviour, we coordinated it with the other behaviours that receive immediate reinforcement such as edge following and goal seeking and we learnt the best MF for each behaviour and the coordination parameters between the different behaviours to learn a good HFLC [11, 12, 13, 14, 15,16]. We then applied our online learning for adaptation and the life long learning techniques to the outdoor robots navigating in changing environments. During our experiments in the indoor environments we tried to emulate a dynamically changing environment by deliberately destroying rules in behaviours or by changing some state of the environment/robot (e.g. loosening a wheel) causing the robot to collide with obstacles, or deviate from the desired response.

After developing a robotic controller on the prototype robot (indoors), the same controller was then transferred to the target (outdoor) vehicles which then adapted the controller to its specific needs using the online adaptation module which is then used for life long learning. Thus the method is robot independent and can be ported between different robots with minimum changes. Figure (11-a) shows the outdoor electrical robot modifying its controller which consists of three coordinated behaviours which are left and right edge following and obstacle avoidance to suit the outdoor environment and the robot kinematics. The robot learns the necessary adapted rules in 79 seconds (using a 68040 microprocesor and a robot of maximum speed 1m/sec) following an irregular metallic fence in outdoor environment with an average deviation of 3 cm and a standard deviation of 1. Figure (11-b) shows the robot coordinating 4 behaviours (left and right edge following and obstacle avoidance and goal seeking) before adapting to the outdoor environments following the center line of an outdoor metallic corridor and trying to achieve a goal at the end of this corridor displaying large deviations of 30 cm with a standard deviation of 9. Figure (11-c) shows the robot after adaptation following a center line with an average deviation of 2.1 cm and a standard deviation of 0.8 under different environmental conditions such as rain, wind, sunshine and starting from different positions. Figure (11-d) shows the outdoor electrical robot coordinating three behaviours which are the left/right edge-following and the obstacle avoidance behaviour to follow the center line of the corridor while avoiding closely spaced objects.

In Figure (12-a) the robot adapts its controller learnt in the indoor prototype robots composed of two coordinated behaviours which are right edge following and obstacle avoidance to follow an irregular metallic fence full of bumps in an outdoor environment at a desired distance of 120 cm whilst avoiding obstacles at a distance of 1m. The robot converged to a solution after an average of 6 iterations taking 3 minutes of robot time (this robot is faster than the indoor robot). It gave a small average deviation of 1.8 cm and a standard deviation of 0.64 for the edge following, and average deviation of 2.4 cm and standard deviation of 0.52 for the obstacle avoidance safe distance.

In Figure (12-b) the robot was applied to follow an edge with a lot of obstacles of different sizes. The robot again produced a smooth response with small average and standard deviations for edge following and obstacle avoidance. Figure (12-c) shows the electrical outdoor robot after adaptation (it is coordinating two behaviours which are right edge following and obstacle avoidance) following an irregular crop edge in outdoor changing and dynamic environment with an average deviation of 3 cm and a standard deviation of 1. Figure (12-d) shows the diesel powered robot with different sensors (mechanical wands to sense the distance from the crop) and different physical and geometrical appearance. The robot applied the same algorithm and used the indoor controller, adapting two behaviours which are right edge following and obstacle avoidance to its own needs and following an irregular hay crop (full of gaps) with an average deviation of 2.4 cm and a standard deviation of 1.

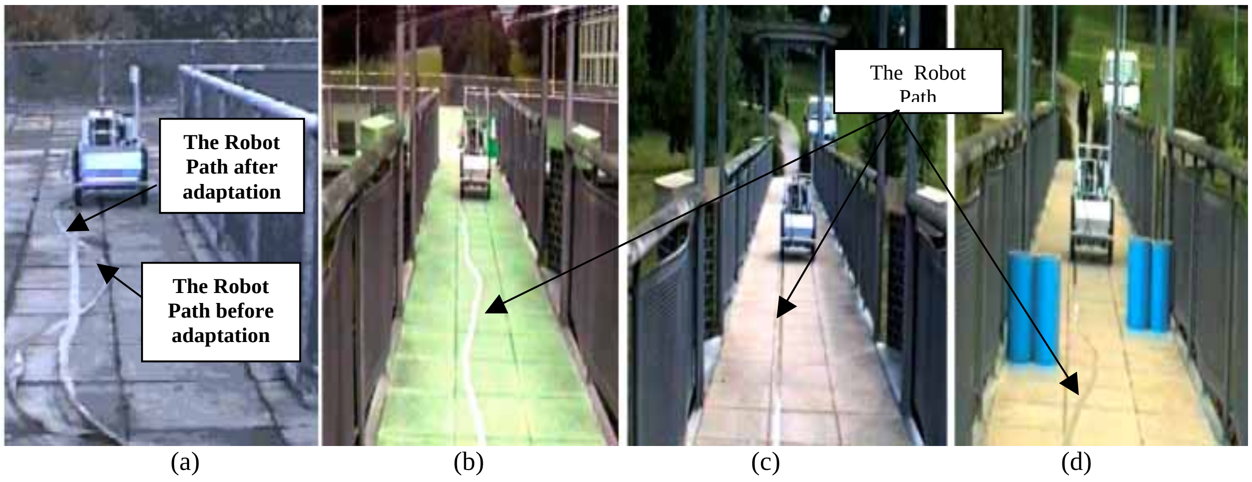

The Robot Path after adaptation
The Robot Path before adaptation
The Robot Path
(a)
(b)
(c)
(d)

Figure (11): a) Electrical outdoor robot path following an irregular edge before and after adaptation in outdoor environment. b) Electrical outdoor robot path before adaptation in an outdoor corridor. c) Electrical outdoor robot path after adaptation in an outdoor corridor. d) Electrical outdoor robot after adaptation navigating in a complex corridor.

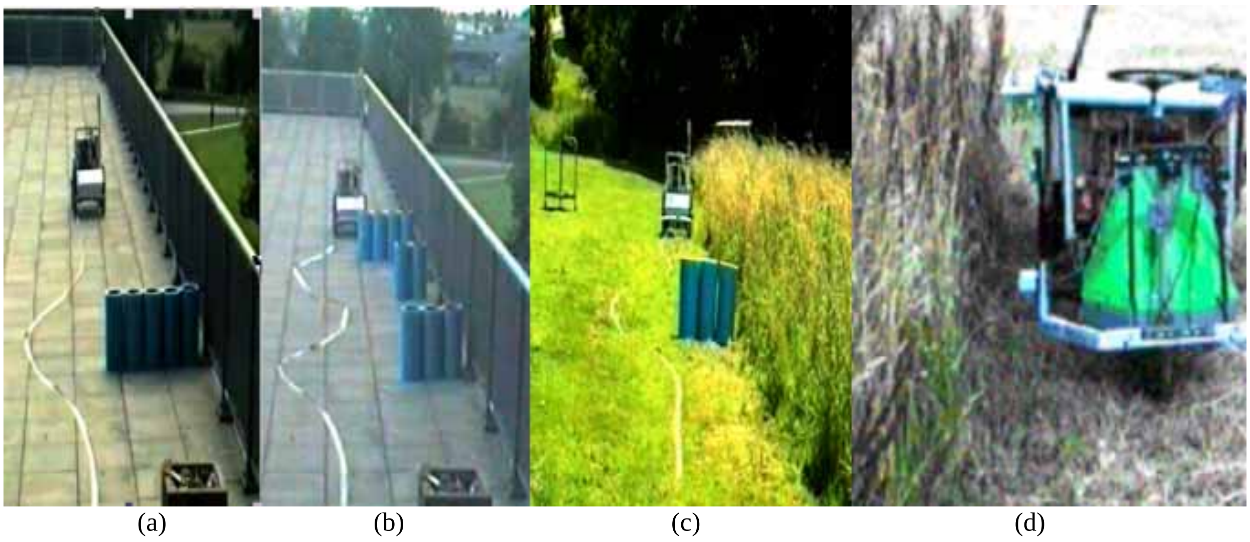


Figure (12): a) The outdoor electrical robot adapting its behaviours to follow an irregular edge while avoiding obstacle. b) The adapted behaviours applied different sizes of obstacles. c) The adapted behaviours applied to follow an irregular crop edge while avoiding obstacles. d) The diesel outdoor robot following an irregular crop edge after adaptation.

5. Related Work

In this section we will introduce related work and how our work is compared to the other work.

Mattellan et al [28] have designed a GA to optimize populations of rule-bases (rather than populations of rules as in my case). Their system had a stopping criteria after a specified length of iteration and not after satisfactory performance as in our case. Their system consisted of 100 generations of 100 individual and an individual life-time of 20 seconds. Thus they needed 333.33 minutes to converge, while we needed only 25 minutes to have a stable general rule-base while operating online without simulation. In [44], he also had a problem with his objective function when he came to a situation where an obstacle could be avoided from both directions. The robot just chose an arbitrary direction possibly conflicting when coordinated with other behaviours and producing longer paths. He used a mixture of supervised and unsupervised learning that converged after 1000 iterations (compared to maximum of 44 iterations used by my system).

There have been other researchers that tried developing autonomous agents (robots) that learns the robot fuzzy controllers using reinforcement learning. [43] needed 350 iterations to learn a controller in simulation. Also [9] used a modified R-learning method to learn the obstacle avoidance behaviour which needed 10000 iterations in simulation. These times are very slow and thus cannot be applied to learn online like our system.

There are other methods that uses Neuro-Fuzzy systems like [42] who developed a self adaptive neuro-fuzzy system and he used simulations to develop a controller for underwater vehicles which needs training data. In case of online learning using real robots this training data will not be available. Also [22] developed a system for evolving connectionist and fuzzy system which he applied for online learning of some problems like speech recognition, but the problem of his system that it might not be appropriate to autonomous embedded agents as his method needs many trials before achieving a satisfactory result. So we conclude that our method has produced a fast covering algorithm that can learn the fuzzy rule-base of the individual behaviours online and through interaction with the environment using real robots.

There have been some work on life-long learning by [30] but it is only limited to operating the robot for long times indoors to build a map of its environment not to learn and adapt itself in a short time interval in outdoor environment and to use its experience to solve situations like our system.

Our method has performed much better than the other methods that were implemented as simulations, using faster processors. Also in our system the agent learns along its life and gains experience using our online adaptation module which constructs the rule-base by learning different situations (local solutions) followed by the interaction among local models. Due to the intersection of neighboring 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 [28], [44].

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 technique are based on a newly patented double hierarchical Fuzzy-Genetic system which are able to learn and adapt complex behaviours of intelligent robots in a short time interval with no human intervention and implementing life long learning. We focused in this paper on learning 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) followed by the interaction among local models. Due to the intersection of neighboring 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 the *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 used *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. Also we used adaptive GA parameters in addition to simple but effective ending criteria. All of the algorithm parameters are robot size independent, not designed for a specific kind of robots and thus being easily moved between differing robots. All of these techniques have resulted in fast convergence, learning the desired behavior online via interaction with the environment in a relatively short time (bounded by the robot speed). The learnt rule-base also has the ability to generalize when moved to other geometrical shapes not encountered during training. Also the system optimized the rule-base and reduced the number of rules. The system is flexible and can add rules or delete them in the case of changing environment or robot kinematics.

We combined the learnt rule base for the obstacle avoidance with the rule bases for the other behaviours learnt in our previous work [11,12,13,14], we also learnt the best membership function for each behaviour and the coordination parameter as reported in our previous work [15, 16].

After developing a good controller in the real robot or its prototype, the controller is then moved to the big outdoor robot operating in its changing unstructured environment. The robot then runs a fast adaptation procedure to adapt the learnt controller to the changing kinematics and environment, and implementing a life-long learning scheme where the agent gains experience through its life time through interaction with the real environment that helps it to get smarter and to solve problems with no need for retraining or human intervention. Our techniques had been verified in difficult domains which are difficult to be solved using the current learning and adaptation techniques like outdoor robots navigating in unstructured environments (such as the agricultural environments). The principal advantages of using a prototype robot is that it avoids the problems associated with software simulation and it learns controllers for the outdoor robots which might involve dangerous maneuvers (like helicopters and underwater vehicles) using small and cheap prototype robots in a controlled environment, thus avoiding the risks associated with online learning using heavy and expensive robots.

We have satisfied the definition of Intelligent Autonomous robotic agent system stated in Section (1.1) which to the author’s knowledge no other work in the literature had satisfied in the field of robotic agents operating in changing unstructured environments such as outdoor agricultural environments. Also our learning techniques learn general controllers that can be applied to different robots performing the same mission and applying a short adaptation cycle thus saving the need to learn a new controller for each different robot.

Using our hierarchical learning procedure has the following advantages: the system proceeds by learning general behaviours, which can then be combined in different ways to achieve different goals, also we have a simplified design procedure as a complex problem is decomposed into a set simpler ones. These results in a fast online autonomous learning system for real-world based physical robots which is, as far as we can determine, a unique achievement [Hagras 2000a, Hagras 2000b]. Also the fuzzy controller is based on a hierarchical fuzzy controller, which have the following advantages:

- It simplifies the design of the robotic controller and reduces the number of rules to be determined.
- It uses the benefits of fuzzy logic to deal with imprecision and uncertainty.
- It uses fuzzy logic for the co-ordination which provides a smooth transition between behaviours with a consequent smooth output response.
- It offers a flexible structure where new behaviours can be added or modified easily.

Also the learning system is based on a double hierarchical fuzzy-genetic system which has the following advantages:

- The hierarchy preserves the system experience, and speeds up the genetic search as the GA is started from the best known point in the search space.

- It rapidly produces good (but sub-optimal) solutions for the individual behaviours and the co-ordination parameters faster than any other reported method utilising real agent in the literature as explained in Section (5).
- Also our system learns rules when it needs, thus our system does not require the robot to learn all the rules to fill up the rule base, but it learns rules that are required to operate the agent in its environment thus leading to rule reduction. Rules can be added, deleted, modified according to the agent requirements.

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.

Although we had applied our system to changing environments these environments were slowly changing such as the agricultural environment and intelligent buildings [6, 18]. For our current and future work we intend to expand our techniques so that we will deal with the fast changing environments such as the flying robots [17] and helicopters and underwater vehicles.

References

[1] H. Beom, H. Cho, “ A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning”, IEEE transactions on systems, man, cybernatics, Vol. 25, No. 3, (1995) 464- 477.

[2] A.Bonarini, F. Basso, "Learning to Co-ordinate Fuzzy Behaviours for Autonomous Agents", International Journal of Approximate Reasoning, special issue on genetic-fuzzy systems for control and robotics, Vol. 17, No. 4, (1997) 409-432.

[3] A. Bonarini, "Comparing Reinforcement Learning Algorithms Applied to Crisp and Fuzzy Learning Classifier systems", Proceedings of the Genetic and Evolutionary Computation Conference, (Orlando, Florida, July, 1999) 52-60.

[4] J. Borestein, Y. Koren, “ Real time obstacle avoidance for fast mobile robot”, IEEE transactions on Systems, Man, Cybernetics, Vol. 19 (October 1989) 1179-1187.

[5] R. Brooks, "Artificial Life and Real Robots", (MIT press 1992).

[6] V.Callaghan, , G.Clarke, M.Colley , H.Hagras, “A Soft-Computing based DAI Architecture for Intelligent Buildings” to appear in the book series entitled Studies in Fuzziness and Soft Computing , (Springer Verlag, August 2000).

[7] M. Delgado, F. Zuben, F. Gomide, “ Evolutionary design of Takagi-Sugeno Fuzzy Systems: a Modular and Hierarchical Approach”, Proceedings of the 2000 IEEE International Conference on Fuzzy Systems, San Antonio-USA, May 2000.

[8] M. Dorigo, M. Colombetti, "Robot Shaping: Developing Autonomous agents through learning", Artificial Intelligence Journal, Vol.71, (1995) 321-370.

[9] G. Faria, R. Romero, “ Incorporating Fuzzy Logic to Reinforcement Learning”, Proceedings of the 2000 International Conference on Fuzzy Systems, (San Antonio-USA, May 2000).

[10] T. Fukuda, N. Kubota , "An Intelligent Robotic system based on Fuzzy approach", Proceedings of the IEEE, Vol. 87, No. 9, (September 1999) 1448-1470.

[11] H. Hagras, V. Callaghan, M. Colley, “A Fuzzy-Genetic Based Embedded-Agent Approach to Learning and Control in Agricultural Autonomous Vehicles”, Proceedings of the 1999 IEEE International Conference on Robotics and Automation, (Detroit- U.S.A, May 1999) 1005-1010.

[12] H. Hagras, V. Callaghan, M. Colley, “Online Learning of Fuzzy Behaviours using Genetic Algorithms & Real-Time Interaction with the Environment”, Proceedings of the 1999 IEEE International Conference on Fuzzy Systems, (Seoul-Korea August 1999) 668-672.

[13] H. Hagras, V. Callgahan, M. Colley, "An Embedded-Agent Architecture for Online Learning and Control in Intelligent Machines", in "New Frontiers in Computational Intelligence and its applications", (Eds. M. Mohammadin, IOS press 1999) 165-174.

[14] H. Hagras, V. Callgahan, M. Colley, "An Embedded-Agent Technique for Industrial Control Environments Where Process Modeling is Difficult", The International Journal of Assembly Automation, (November 1999) 323-331.

[15] H. Hagras, V. Callaghan, M. Colley, “On Line Calibration of the sensors Fuzzy Membership Functions in Autonomous Mobile Robots” to appear in the Proceedings of the 2000 IEEE International Conference on Robotics and Automation, (San Francisco-USA, April 2000) 3233-3238.

[16] 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 ” to be appear in the Proceedings of the 2000 IEEE International Conference on Fuzzy Systems, (San Antonio-USA, May 2000) 853-859.

[17] H.Hagras, V.Callaghan, M.Colley, "Learning and adaptation for flying robots", Invited paper in the third international conference in non-linear problems in Aviation (Daytona Beach, Florida-USA, May 2000).

[18] H.Hagras, V.Callaghan, M.Colley, G.Clarke, “A Hierarchical Fuzzy Genetic Multi-Agent Architecture for Intelligent Buildings Sensing and Control ”, In the International Conference on Recent Advances in Soft Computing 2000 (Leicster-UK June 2000).

[19] F. Hoffmann, "Incremental Tuning of Fuzzy Controllers by means of Evolution Strategy", Proceedings of the GP-98 Conference, (Madison, Wisconsin, 1998) 550-556

[20] L. Kaelbing, M. Littman, “ Reinforcement Learning: A survey”, Journal of Artificial Intelligence Research, vol. 4 (1996) 237-285.

[21] N. Kasabov, “Introduction: Hybrid intelligent adaptive systems. International Journal of Intelligent Systems” Vol.6, (1998) 453-454.

[22] N. Kasabov, M. Watts, " Neuro-Genetic Information Processing for Optimisation and Adaptation in Intelligent Systems", In: *Neuro-Fuzzy Techniques for Intelligent Information Processing*, N.Kasabov and R. Kozma, (Heidelberg Physica Verlag 1999) 97-110.
[23] O. Khatib, " Real time obstacle avoidance for manipulators and mobile robots", International Journal of Robotics research, Vol.5, (1986) 90-98.
[24] C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller, Part I", IEEE transactions on Systems, Man, Cybernetics, Vol.20, (March 1990) 404-432.
[25] C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller, Part II", IEEE transactions on Systems, Man, Cybernetics, Vol.20, (March 1990) 419-434.
[26] D. Leitch, "A New Genetic Algorithm for the Evolution of Fuzzy System", Ph.D. thesis, University of Oxford 1995.
[27] G. Linkens, O. Nyongeso, "Genetic Algorithms for Fuzzy Control, Part II: Online System Development and Application", IEE proceedings Control theory applications, Vol.142 (1995) 177-185.
[28] V. Matellan, C. Fernandez, J. Molina, "Genetic Learning for Fuzzy Reactive Controllers", Journal of Robotics and Autonomous Systems", Vol. 25, (1998) 33-41.
[29] 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.
[30] U. Nehmzow, " Continuos Operation and perpetual Learning in Mobile Robots", Proceedings of the International Workshop on Recent Advances in Mobile Robots, (Leicester-UK June 2000).
[31] P.Pal, A. Kar, " Mobile robot navigation using a neural network", Proceedings of the IEEE International Conference on Robotics and Automation, (1995) 1503-1508.
[32] M. Rocha, P. Cortez, J. Neves, " The Relationship between Learning and Evolution in Static and Dynamic Environments", Proceedings of the Second ICSC Symposium on Engineering of Intelligent Systems, Paisley-UK, (June 2000) 500-506.
[33] E. Ruspini, "Truth as Utility a Conceptual Synthesis", In Proceedings of the 7th Conference on uncertainty in Artificial Intelligence, (Los Angeles-USA) pp. 458-464,
[34] A. Saffiotti, "Fuzzy Logic in Autonomous Robotics: Behaviour Co-ordination", Proceedings of the 6th IEEE International Conference on Fuzzy Systems, Vol.1, (Barcelona, Spain, 1997) 573-578.
[35] A. Schwartz, " A reinforcement learning method for maximising undercounted rewards," in Machine learning: Proceedings of the 10th International Conference, San Mateo, CA, Morgan Kaufmann, (1993) 450-457.
[36] M. Sousa, M. Madrid, " Optimisation of Takagi-Sugeno Fuzzy Controllers Using a Genetic Algorithm", Proceedings of the 2000 IEEE International Conference on Fuzzy Systems, (San Antonio-USA, May 2000).
[37] M. Srinivas, L. Patnaik, "Adaptation in Genetic Algorithms", Genetic Algorithms for pattern recognition, Eds, S.Pal, P.Wang, (CRC press 1996) 45-64.
[38] L. Steels, "When are Robots Intelligent Autonomous Agents", Journal of Robotics and Autonomous Systems, Vol. 15, (1995) 3-9.
[39] G. Tan, X. Hu, "More on Designing Fuzzy Controllers Using Genetic Algorithm: Guided Constrained optimisation", Proceedings of the 6th IEEE International Conference on Fuzzy systems, (Barcelona- Spain 97) 497-502.
[40] E. Tunstel, T. Lippincott, M. Jamshidi, "Behaviour Hierarchy for Autonomous Mobile Robots: Fuzzy Behaviour Modulation and Evolution", International Journal of Intelligent Automation and soft computing, Vol.3, No.1, (1997) 37-49.
[41] S. Vijayakumar, S. Schaal, " Fast and Efficient incremental Learning for High-dimensional Movement Systems", Proceedings of the 2000 IEEE International Conference on robotics and Automation, (San Francisco-USA April 2000) 1894-1899.
[42] J. Wang, J. Yuh, "Self-Adaptive Neuro-Fuzzy Systems with fast parameter learning for autonomous underwater vehicle control", Proceedings of the 2000 IEEE International Conference on Robotics and Automation, (San Francisco-USA April 2000) 3861-3866.
[43] N. Yung, C. Ye, " An Intelligent Mobile Vehicle navigator based on Fuzzy Logic and Reinforcement Learning", IEEE transactions on systems, man, cybernatics, Vol. 29, No. 2, (1999) pp. 314- 321.
[44] L. Zhang, V. Schwert, " Rapid Learning of Sensor-Based Behaviours of Mobile Robots Based on B-Spline Fuzzy Controllers", Proceedings of the 1998 IEEE International Conference On Fuzzy Systems, (Anchorage, U.S.A, 1998) 1346-1352.
[45] H. Zhou, " A Computational Model of Cumulative Learning", Machine learning Journal, (1990) 383-406.