Using On-line Genetic Algorithms for Learning and Adaptation Of Hierarchical Fuzzy Behaviors Of a Real Time Mobile Reactive Robot

Hani Hagras, Victor Callaghan, Martin Colley

Abstract-- **This paper introduces a learning algorithm to on-line adapt a robot controller consisting of fuzzy behaviors which are organized hierarchically and coordinated using fuzzy logic to any environmental or ground or even robot dynamics changes. This allows the robot to deal adaptively with outdoor changing environments such as the agricultural environment.**

A modified version of the Fuzzy Classifier system (FCS) is used in this algorithm. The FCS is equipped with Long Term Memory (LTM) to make it possible for the learning system to transfer its problem solving expertise into a solution for the problem of interest, as well as allowing the GA to start its search from the best point found. The system also uses its sensory information in-order to narrow the search space for the Genetic Algorithm (GA). Adaptive mutation is also used to speed up the GA search. The proposed techniques have resulted in a fast converging algorithm that can be applied to adapt as well as learn the robot behaviors to perform a global task (such as get out of a maze while avoiding obstacles) on-line with real robots with no need to simulation. The proposed system is also characterized by being adaptive so that if any of the environmental conditions or the robot dynamics is changed the robot can still adapt itself to the environment without the need to repeat the learning cycle from the beginning. The algorithm is robot independent so that it can be applied to different robots irrespective of their shapes or sizes. The Results achieved with a real robot are discussed and compared with the other methods to show the effectiveness and the speed of the proposed method.

Index Terms-- **Fuzzy Logic, genetic algorithms, classifier systems, mobile robots.**

I. INTRODUCTION

In the past several years, fuzzy logic control has been explored for mobile robot reactive navigation [22] [24]. A robot control system is decomposed into several task oriented parallel computing modules called behaviors [21]. Each behavior is implemented with a set of fuzzy control rules, which has the form if *x is A and y is B then z is C.*

Through fuzzification, fuzzy set operations and fuzzy reasoning processes, a fuzzy control rule produces a control output. The control outputs of a fuzzy behaviors are produced by synthesizing all the outputs of the fuzzy control rules through defuzzification. The main advantage of fuzzy logic control is that expert knowledge and human experiences can be easily translated into fuzzy control rules. A fuzzy logic controller is also capable of accommodating approximate, imperfect and noisy information presented in real world environments and producing smooth control output [24].

GA are a stochastic global search method that mimic the metaphor of natural biological evolution. GA operated on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their fitness in the problem domain and breeding them together using operators borrowed from natural genetics. The GA begins by initialization of the genes of each individual in the population *P(k),* where *k* is the number of generations. It then generates *P(k+1)* from *P(k)* by evaluating the fitness of each individual in *P(k)* and selection of individuals from *P(k)* with a probability proportional to their fitness. Recombine, reproduce and mutate them using the genetic operators of reproduction, crossover and mutation. If termination condition is met, stop and return the best individual. Otherwise set *k=k+1* and produce new *P(k)* [8].

The goal of our research is to develop a robot for the outdoor agricultural domain. In an agricultural setting the inconsistency of the terrain, the irregularity of the product and the open nature of the working environment result in complex problems of identification and sensing and control. Problems can range from the effects of varying weather conditions on vehicle sensors and traction performance, through to the need to deal with the presence of unauthorized people and animals. All these problems provide good opportunities for fuzzy systems as they excel in dealing with imprecise and varying conditions which characterizes such situations. In previous work [10], we have used a hierarchical fuzzy control architecture for controlling the robot in an outdoor agriculture media, but the parameters of the fuzzy controllers must be varied under different field environmental changes and robot kinematics changes. So for successful out-door navigation we need a

The Computer Science Department, Essex University, Wivenhoe Park , Colchester CO43SQ, England, U.K .

fast learning and adaptive system. In this paper a new algorithm is developed which is using modified version of FCS which have its GA search space reduced by the sensor data. Also this algorithm is equipped with a long term memory so that to make it possible for the learning system to transfer its problem solving expertise into a solution for the problem of interest, as well as allowing the GA to start its search from the best point found to the moment in the search space instead of starting from scratch. Also the practical problems involved with using learning for real robots such as determining the distance traveled by the robot (to be used in the objective function) are solved. The algorithm is designed to be robot independent so that the algorithm can be applied to different robots independent of their shapes or sizes. This algorithm will be used to modify and learn rules of a robot controller consisting of fuzzy behaviors which are organized hierarchically and coordinated using fuzzy logic to perform a global task (such as get out of a maze while avoiding obstacles) on-line with real robots with no need to simulation. The algorithm is adaptive to any environmental or ground or even robot dynamics changes, with no need to repeat the learning cycle whenever any of the surrounding circumstances changes.

The proposed system can be viewed as a double hierarchy system in which the fuzzy behaviors are organized in a hierarchical form and the online learning algorithm is also a hierarchy in which in the higher level we have a population of solutions stored in the LTM 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 by finding the distance it had moved before failing. The highest fitness experience is used as a starting position to the lower level GA which is used to produce new solution to the current situation.

This paper is organized as follows. In the next three section we briefly introduce the merits of learning using real robots over learning by simulation then we introduce the work done in designing fuzzy controllers using GA. Then fuzzy classifier systems and fuzzy hierarchical controller are introduced. Then we outline the algorithm and outline the problem definition and then explain the algorithm different components and the LTM technique and the choosing of crossover and mutation techniques and their variation effect on convergence rate. In the final section we introduce the results of the experiments done on real robots and compare them with results obtained by the researchers in this field.

A. *Why Online Learning*

Broadly speaking, our work situates itself in the recent line of research which concentrates on the realization 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 activity in the real world in real time. Another important point is that adaptive behavior 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[5].

Despite most robotics regularly use simulations to test their models, the validity of computer simulations to build autonomous robots is criticized and the subject of much debate. Computer simulations may be very helpful to train and test robotics models. However as Brooks[4] 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 who want to use computer models to develop control systems for real robots may encounter problems [19]:

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 very uncertain effects, whereas simulative models often use grid-worlds and sensors which return perfect information.
c) Different physical sensors and actuators, even if apparently identical, may perform differently because of slight differences in the electronics and mechanics or because of their different positions on the robot.

Even if some researchers are using real robots to learn behaviors, these behaviors if learnt successfully are usually frozen in the robot so that if some of the robot dynamics is changed or the environmental circumstances is changed , the robot must repeat a time-consuming learning cycle.

In our case we aim to use Fuzzy Classifier Systems with GA as a rule discovery system to adapt the robot to ongoing environmental changes. Such adaptivity to the environment is important especially if using outdoor agricultural robots where the agricultural environment is rapidly changing.

B. *Fuzzy Logic and GA Learning*

In many applications the robot's environment changes with time in a way that is not predictable by the designer in advance. In addition , the information available about the environment is subject to imprecision , incompleteness and imperfection due to the perceptual quality of sensors. These problems limits the utility of traditional model-based reasoning approaches.

Evolutionary algorithms constitute a class of search and optimization methods guided by the principles of natural evolution. GA are optimization methods inspired by principles of natural evolution and genetics. 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. Their capability of learning enables a GA to adapt to a system to deal with any desired task .

Fuzzy logic offers a framework for representing imprecise , uncertain knowledge. Similar to the way in which human beings make their decisions fuzzy systems are using a mode of approximate reasoning, which allows them to deal with vagueness and incomplete information. Fuzzy controllers show robustness with regard to noise and variations of system parameters.

Combinations of various soft computing disciplines which includes fuzzy logic ,neural networks genetic algorithms have acquired the name of *hybrid* systems. Several work have focused in automating the design of the rule bases so that the fuzzy controller is fully optimized.

Karr [14] , developed systems that learned to balance an inverted pendulum. The system learnt slowly , taking several thousand generations to develop a good controller.

Lee and Takagi developed the technique in [16] , using real coded GA to represent rules as vectors. They used a fixed length string and a variable length string , with results markedly better than Karr's on the same problem ,due to the fact that the GA had the flexibility to design the system antecedent sets.

Herrera et.al [11] describe a system where the fuzzy rule bases are coded using strings of real numbers , combined with arithmetical crossover and mutation operators.

Leitch in [17] had developed a new algorithms in which he used a new coding technique called context dependent coding (CDC) which is unlike the position dependent schemes where the meaning of the codon is determined by its absolute position in a chromosome. The CDC codon's interpretation is determined by the context in which it is , that is the meaning is dependent on the values of surrounding codons. This means that some sequence of codons will have the same interpretation regardless of where they lie on the chromosome. The main advantage of this is that it allows for great flexibility , so crossover is very simple and can occur at any site as the coding is robust to disruption due to the meaning being dependent on context rather than position. He also used an implicit chromosome reordering operator which improved the algorithm performance for this application , reducing epistasis by using an estimate of it to favor chromosomes with low epistasis during selection. Because he was using simulation, he introduced the co-evolution of controller test sets which leads to a situation similar to a biological predator/prey pair , where controllers are continually adapting to a new test sets , while the test sets adapt to be as difficult for the controller as possible. He used the GA off-line using simulation and training data set to optimize his fuzzy rule set in robotics for very simple independent problems like corridor tracking , performing a multi point turn in a confined space.

Hoffmann [12] had implemented a new design of hierarchical fuzzy controllers using messy genetic algorithms which is unlike the classical GA which encode candidate solutions to strings of fixed length. Messy GA work with strings of flexible length in which genes can be arranged in any order. Each gene is composed of a pair of integers. The first entry specifies the meaning of the gene , which in case of a standard coding is determined by the location within the string. The second integer plays the same role as in classical GA by representing the value of the gene. This algorithms has new genetic operator such as the cut-splice operator which replaces the crossover. It is immune to disruption by the crossover techniques and very robust as the chromosome is variable sized. He also applied the problem again to the simulation of learning the obstacle avoidance and goal seeking for the robot by using different input data and applied the controller to a real robot.

However a lot of work remains to be done. The most important problems are to increase the speed of convergence while maintaining stability, implementing pure reinforcement learning , these problems will be involved in our research.

C. *Fuzzy Hierarchical Systems*

Modular decomposition is a well known technique for reducing system complexities. Hierarchies are a proven method of effectively handling and managing modularized control structures . Albus[1], among others shows how any complex activity can be decomposed into a hierarchy of behavioral modules each consisting of few behaviors. He sites many examples of such systems ranging from the organization of government through to control architecture.

For mobile robot and complex reactive systems, the size of the input space requires a complicated control function. This mapping can be made manageable by breaking down the input space for analysis by multiple agents, each of which responds to specific types of situations and then integrating the recommendations of these agents. Agents also called behaviors, can be designed independently to exhibit behaviors such as goal seeking, obstacle avoidance, and wall following [24]. The work presented in this paper seeks to apply these hierarchical efficiencies to the organization of fuzzy architectures.

There are many ways for behavior co-ordination. A classical robot architectures such as the subsumption[4] architectures which decomposes the system into small independent decision-making processes , or behaviors. These architectures use a on-off switching schema : in each situation , one behavior is selected and is given complete control of the effectors. This simple scheme may be inadequate in situations where several criteria should taken into account. Also this rigid organization contrasts with the requirement that an autonomous robot can be programmed to perform a variety of different tasks in a variety of environments[24]. Later proposals relied on dynamic arbitration policies , where the decision of which behavior to activate depends on both the current (sub- goal), given by the planner and the environmental conditions. Both fixed and dynamic arbitration policies can be implemented using the mechanisms of fuzzy logic. The two main advantages in doing so are :

a) The ability to express partial and concurrent activation's of behaviors .
b) The smooth transition between behaviors[9].

In previous work [10] we have developed a fuzzy hierarchical controller which can combine four behaviors and navigate in an unknown environment reactively .

D. *Fuzzy Classifier Systems*

A classifier system in an adaptive, general purpose machine learning system which is designed to operate in noisy environments with infrequent and often incomplete feedback.

Classifiers simply are if-then rules. The name Learning Classifier Systems (LCS) comes from the capability of rules to classify messages into arbitrary message sets [13]. However, this is only one facet of rules. In classifier systems rules or productions have the same role as instructions in ordinary programs. Such production systems are computationally complete [20] and therefore as powerful as any other Turing-equivalent programming language.

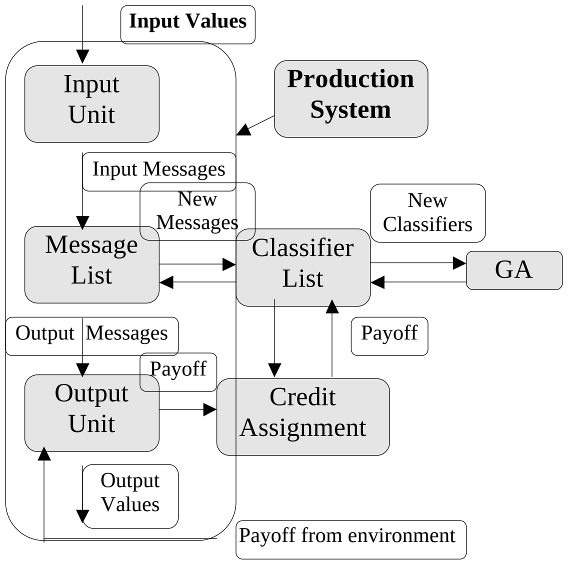


Figure 1 : The classifier system.

A classifier systems is a machine learning system which learn rules in order to guide its performance in any arbitrary environment [7]. Its main components are a production system and one or several learning algorithms as shown in Fig. 1. This classifier system is characterized by a very simple pattern language, parallel rule firing and message-based internal communication.

At the top level, the classifier system communicate with the environment. The classifier system effects action in the environment and detects information on the state of the environment. Moreover, an action or sequence of actions may lead to payoff received by the classifier system. The classifier system consists of the production system and two learning components, namely an apportionment of credit system and a rule discovery system. In a Holland classifier system [13], the apportionment of credit algorithm for updating rule weights is a bucket brigade algorithm, the rule discovery algorithm is a GA.

The bucket brigade algorithm modifies the weight of rules (called the strength) in the rule base with the payoff from the environment, with payments from message consuming rules and with payments to message producing rules. However, for the bucket brigade to work properly, all useful rules must be present in the rule base [5].

Generating new rules is the task of the GA. The GA sees the rule base as a population of classifiers, whose fitness is the rule-strength obtained under the bucket brigade algorithm. The GA is invoked by the production system periodically, and it generates a new population of rules according to rule-strength. However for the GA to work properly, the strengths of the rules generated by the bucket brigade algorithm must reflect the true fitness of the rules.

There are two different approaches in learning fuzzy controllers using GA. In the so called “Michigan” approach of GA [7] the population consists of fuzzy rules. The fitness is assigned to individual rules competing among each other in the evolution process. This approach is appropriate for on-line learning because the fuzzy controller is built of the population itself and is improved constantly in the evolution process [18]. A mechanism of credit assignment to individual rules is required , which is difficult when reinforcement is only provided sporadic after a sequence of control actions. Credit assignment procedures like the bucket brigade can be used to distribute reinforcement among fuzzy rules activated sequentially in time.

The so called “Pitts” approach of GA uses a population of fuzzy controllers. Each individual alone is a candidate solution to the optimization problem. It is only possible to learn off-line because in each generation a population of solutions has to be tested [18]. The fitness function evaluates the performance of the entire fuzzy controller. The assignment of credit is easier but involves the drawback that rules of bad quality sometimes benefit from good ones.

In many complex environments the LCS have not had not much application due in part to the limitations of their syntax to represent continuously varying variables. A simple and promising way of dealing with this problem is through fuzzy set theory [11].

A FCS is a genetic based machine learning system whose classifier list is a fuzzy rule base. They learn by creating fuzzy rules which relate the values of the input variables to internal or output variables. They integrate the same elements of the LCS but working in fuzzy environment.

Valezuela –Rendon [23] gave the first description of the fuzzy classifier system , the classifiers are fuzzy rules , similar to fuzzy controllers. Each classifier is a binary string that encodes the membership function of the fuzzy sets defined for variables involved in the problem so that the number of bits in a condition or an action is the number of fuzzy sets defined over a given variable. A “1” indicates that the corresponding fuzzy set is part of the condition or action. He tested his fuzzy classifier system in the identification of static one-input one-output systems using a stimulus-response fuzzy classifier system .

Bonelli [3] produced a new system in which each variable has associated a fuzzy set and i.e. each variable is described by a membership function. This description is variable and will evolve through genetic search. Each classifier contains the actual description of the membership functions that correspond to each input and output variable, which consists of parameters that define the associated fuzzy set. There is also an associated strength to each classifier that indicate its credibility. The degree to which each classifier is activated is calculated by taking the minimum of the current inputs membership values with respect to the fuzzy sets present in the condition part of each classifier. In the output interface the partially activated fuzzy sets of same output variables are combined using the weighted sum method to produce a final fuzzy set for each output variable. Its credit assignment system only works with positive rewards. It deducts a fraction of each active classifier strength and distributes the payoff quantity of the obtained reward to each active classifier strength according to a measure of goodness. This measure determines the quality of the classifiers action and the quality of the classifiers conditions for this particular input. He applied this model to the same examples used by [23] and he obtained better results.

Very few and simple applications of on-line learning in robotics among these are Bonarini [2] in which he suggests a hybrid method solving the co-operation versus competition problem. He uses sub `populations of similar fuzzy rules ,which are undergoing a local competition. Co-operation of fuzzy rules is achieved by composing each of the best local solutions into an entire fuzzy controller, he had applied this method for simple behaviors like following another robot or moving in a corridor and then he had coordinated them, but in this work he developed his controllers by simulation and then he applied it to real robots so it is not pure on-line learning.

Other work was done by Furuhashi [7] on which he based his credits for each rule on the number of membership function in the antecedent having values larger than zero. He applied his algorithm with a standard GA to a very simple simulation problem of two ships attempting to avoid each other.

II. The Fuzzy Hierarchical System Configuration

Most commercial fuzzy control implementations feature a single layer of inferencing between two or three inputs and one or two outputs. For autonomous robot, however the number of inputs and outputs are usually large and the desired control behaviors are much more complex. For example in our case we have 7 sonar inputs and an infrared bearing sensor i.e. eight inputs and we have two outputs which are the left and right wheel speeds and assuming that each input will be represented only by three fuzzy sets and each output by four fuzzy sets. In this case, using a single layer of inferencing will lead to determining 3^8= 6561 rules which is difficult to determine if not impossible. While if we divide the whole system to four co-operating behaviors , the obstacle avoidance which consists of three sonar inputs each represented by three fuzzy sets, this leads to determine 3^3 =27 rules, the left and right wall following each having two sonar inputs each represented by three fuzzy sets this will lead to 3^2 =9 rules in each behavior, the goal seeking behavior only taking one infrared bearing scanner input each represented by seven fuzzy leading to 7 rules. Then the total required rules to be determined in the individual behaviors are 27+9+9+7=52 rules which is easy to be determined. However we need some form of co-ordination scheme in order to combine these behaviors into a single action. In this paper we have chosen the fuzzy context rule combination method developed by Saffiotti [24] to perform the high level co-ordination between the behaviors. The context depending rules are characterized by each behavior generates *preferences* from the perspective of its goal. Then each behavior has a context of activation, representing the situations where it should be used. The preferences of all behaviors, weighted by the truth value of their contexts, are fused to form a collective preference. Then one command is chosen from the collective preference.

The work described in this paper suggests a solution based on using fuzzy logic to both implement individual behavior elements and necessary arbitration (allowing both fixed and dynamic arbitration policies to be implemented). We achieve this by implementing each behavior as a fuzzy process and then using other fuzzy processes to co-ordinate them.

Each fuzzy process provides some basic machine behavior. In this system four behaviors will be used for robot navigation, namely goal seeking, obstacle avoidance, right edge-following and left edge following.

In the obstacle avoidance behavior, the robot is required to avoid obstacles from the front. To accomplish this task, the three front sensors of the robot are used, which are the Left Front Sensor (LFS), Medium Front Sensor (MFS) and the Right Front Sensor (RFS). The sensor configuration is shown in figure (2).

The left and right edge following are used to follow a wall or an edge on the left or right side of the robot , thus enabling it to navigate out of mazes and in tight corridors. The Left edge following behaviors uses two left side sensors : Left Side Front (LSF), Left Side Back (LSB). The Right edge following behaviors uses two Right side sensors : Right Side Front (RSF), Right Side Back (RSB).

The goal seeking behavior is used for the robot to reach a goal, and the path of the robot to its goal is completely reactive with no previous planning and the goal in our experiments is in the form of an infra-red beacon. To accomplish this behavior the input to this behavior is from an infra-red scanner.

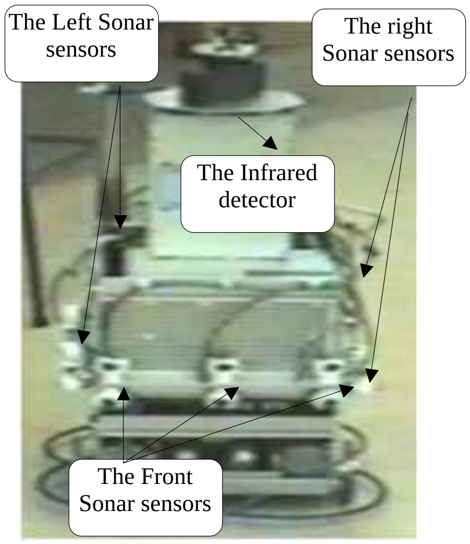


Figure 2 : The robot and its sensor configuration.

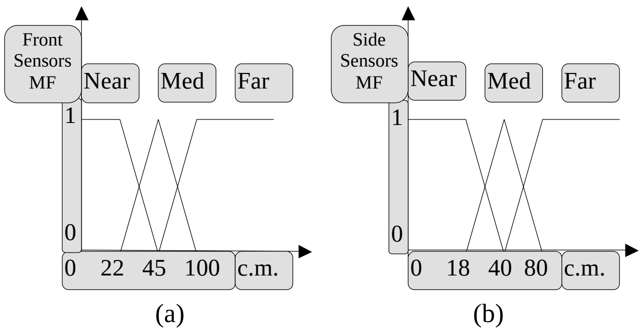


Figure 3: a) The Membership function (MF) of the front sensors b) The MF of the side sensors.

In this work all input Membership Functions (MF) of all the behaviors are pre specified to the controller and are shown in Fig. 3, Fig. 5. The output MF of all the behaviors are shown in Fig. 4-a and are chosen to be the right and left wheel velocities. These MF were used in our previous work to guide a robot in an outdoor agricultural environment and they were derived using human experience according to the designer estimates of the safety distances as well as the upper and lower limits of the sensor readings.

The rule bases of the left and right edge following and the obstacle avoidance are to be learnt and modified on-line using real robots and using the proposed on-line GA algorithm, however we have supplied the rule base of the goal seeking behavior which was designed using human experience. The reason for this is, in this work we are interested in solving the problem of getting out of a maze safely without hitting any obstacles, and we are not interested of finding the optimal path toward the goal, we are only interested in reaching this goal, which is the case in an agricultural domain.

In the following design of each single behaviour we will use singleton fuzzifier, triangular membership functions, product inference, max-product composition, height defuzzification. The selected techniques are chosen due to their computational simplicity.

The equation that maps the system input to output is given by:

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

Where M is the total number of rules , y is the crisp output for each rule, αAi is the product of the membership functions of each rule inputs, G is the number of inputs. More information about fuzzy logic can be found in [15].

The resultant architecture takes a hierarchical tree structure form and is shown in Fig. 6. Saffiotti [24] defines fuzzy command fusion as interpretation of each behaviour producing unit as an agent expressing preferences as to which command to apply. Degrees of preferences are represented by a possibility distribution (or fuzzy as in our case) over the command space. In our hierarchical architecture we use a fuzzy operator to combine the preferences of different behaviour into a collective preference.

According to this view, command fusion is decomposed into two steps: preference combination and decision. In case of using fuzzy numbers for preferences, product-sum combination and height defuzzifcation. The final output equation is [24]:

$$\mathbf{C}= \quad \frac{\sum_i (BW_i * C_i)}{\sum_i BW_i} \qquad (2)$$

Where i = right behavior, left behavior, obstacle avoidance, navigation. C_i is the behavior command output (left and right velocity in our case). These vectors have to be fused in order to produce a single vector C to be applied to the mobile robot. BW_i is the behavior weight. The behavior weights are calculated dynamically taking into account the situation of the mobile robot. By doing this there is no need to pre-plan as the system plans for its self depending on the current situation of the environment.

In figure(6) each behavior is treated as an independent fuzzy controller and then using fuzzy behavior combination we obtain a collective fuzzy output which is then deffuzzified to obtain a final crisp output.

In behavior coordination there are some 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, in this case A= 40 c.m, B=100 c.m. 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, in this case A=18 c.m, B=36 c.m (these values were designed according to the designer interpretation of safe distances) After calculating these values, each of them is matched to its membership function which are shown in Fig. 4-b and 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 d1 IS HIGH AND d2 IS HIGH AND d3 IS HIGH THEN GOAL SEEKING.

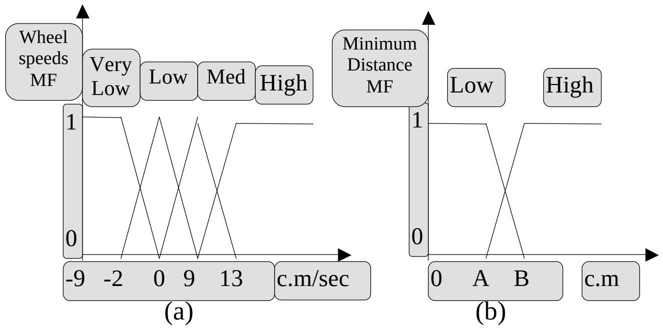


Figure 4: a) The MF of the left and right wheel velocity of the robot. b) The MF of d1,d2,d3.

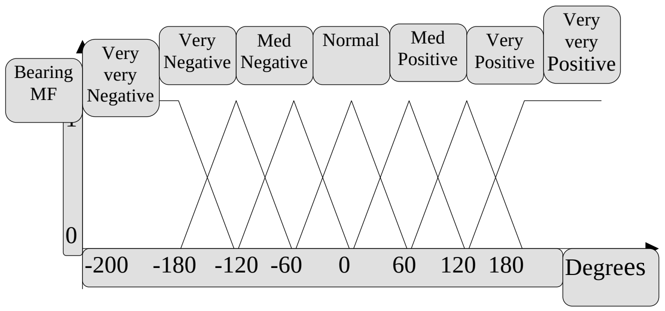


Figure 5: The MF of bearing from the goal for the goal seeking behaviour

These context rules determines which behaviour is fired and to what degree, then the final robot output is calculated using equation 2.

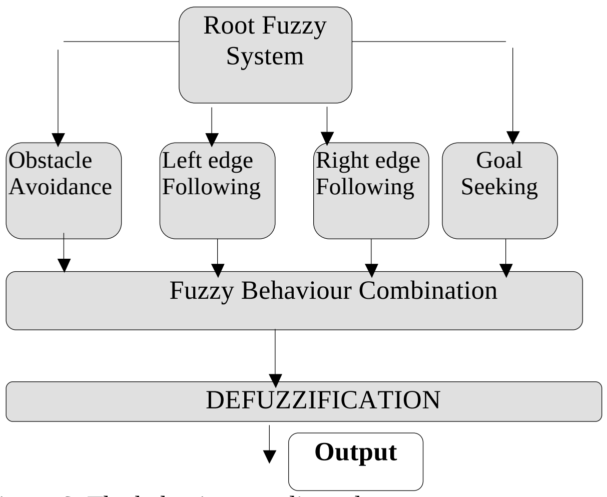


Figure 6: The behavior coordinated system.

III. OVERVIEW OF THE PROPOSED ON-LINE ALGORITHM

In this work, we are considered of making the robot learn to adapt its combined behaviors to achieve a high level goal which is getting out of a maze while avoiding obstacles.

Table 1 : The Fuzzy rule base of the goal seeking behaviour.

Bearing From Goal	Left Velocity	RightVelocity
Very Very Negative	Very Low	Very High
Very Negative	Very Low	Very High
Medium Negative	Low	Medium
Normal	Very High	Very High
Medium Positive	Medium	Low
Very Positive	High	Very Low
Very Very Positive	Very High	Very Low

In a real-time GA, it is desirable to achieve a high level of online performance while, at the same time being capable of reacting rapidly to process changes requiring new actions. Hence it is not necessary to achieve a total convergence of the population to a single string, but rather to maintain a limited amount of exploration and diversity in the population. Incidentally, it can be observed that near-convergence can be achieved in terms of fitness, with diverse structures. These requirements mean that the population size should be kept sufficiently small, so that progression towards near-convergence can be achieved within a relatively short time. Similarly the genetic operators should be used in a way that achieves high-fitness individuals in the population rapidly [18].

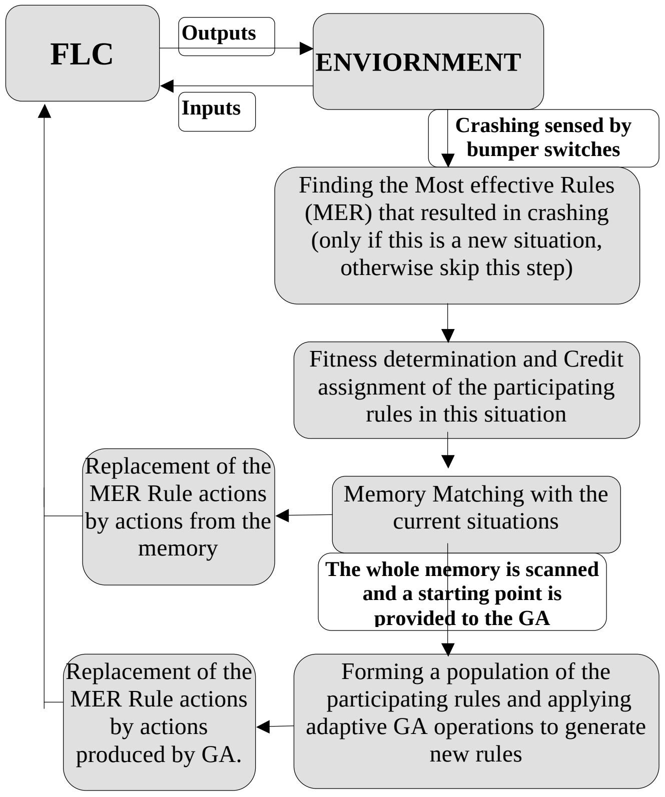

FLC
Outputs
ENVIORNMENT
Inputs
Crashing sensed by bumper switches
Finding the Most effective Rules (MER) that resulted in crashing (only if this is a new situation, otherwise skip this step)
Fitness determination and Credit assignment of the participating rules in this situation
Replacement of the MER Rule actions by actions from the memory
Memory Matching with the current situations
The whole memory is scanned and a starting point is provided to the GA
Replacement of the MER Rule actions by actions produced by GA.
Forming a population of the participating rules and applying adaptive GA operations to generate new rules

Figure 7: Block diagram of the Proposed on-line algorithm.

Fig. 7 introduces a block diagram of the operation of the proposed on-line algorithm. In this system as we are concerned in adapting an existing controller to the existing environment, we will assume that all the behaviors have useless rules that cannot achieve the robot goal (which is getting out of a maze while avoiding obstacles) and need to be modified. This can also be viewed as learning the rules of the different behaviors from scratch. As argued by [21], if the robot starts with a random rule base then it can do unpredictable things which can damage the robot or turn the robot on spot with out moving at all. To avoid this problem, the obstacle avoidance behavior as well as the right and left wall following are randomly initialized to have the same consequent such consequent is move forward with normal speed or move to the right with low speed, but not move with zero speed. By doing this then we make sure that the robot is moving to start its learning sequence. This is similar to classifiers systems where all the classifiers are initialized with same fitness strength. In the following sections the algorithm component will be introduced.

A. *Finding the Blamed Rules*

After rule base initialization of the three behaviors the robot starts moving with these bad rule bases, until it hits an obstacle or wall. Then the on-line algorithm is fired to generate new set of rules to escape from this collision. 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 our 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 [6]. In our case, only 4 rules consequences will be replaced and these rules are the most effective rules in the situation of crashing, because they are the mostly blamed for this crashing. These rules are found by making the robot using its Short Time Memory that maintains the last 2000 actions and replaying them to remember the robot path and find the blamed rules. The distance backed at this step will be used after as the starting point of all the solutions proposed by the algorithm. The method of specifying the rules to be replaced will be explained later in detail.

B. *Fitness determination and Credit assignment:*

The system fitness is evaluated by the distance moved by the robot from its starting point before crashing. To determine this distance we use triangulation between 3 infrared beacons placed at known distances to know how far away is the robot from an origin point. And then by subtracting this distance from the distance of the starting point from the origin we can know the
distance the robot had travelled before crashing.

C. *LTM Application*

After determination of the rules whose consequences to be replaced, the robot then matches the current rules to chunks of rules stored in a LTM. If for examples we have rules 1,2,3,4 to be replaced and in the first chunk we have 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 collision then these rules are kept in the rule base of the controller and we have saved the process of learning a solution to this problem from the beginning by using our memorized experience. If the robot crashes again, it returns to the first point where it had started and measure the distance it had moved to determine the fitness of the solution proposed by this memory chunks. After all memory chunks have been examined and the robot still crashes, the best solution proposed by LTM 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. This LTM will serve to speed up the search.

D. *Producing New Solution by GA*

The GA then starts its search for a new rule consequences for the blamed rules. The fitness of every rule in the population is proportional to its contribution in the final action. If the proposed action by the new solution results in improvement in the distance then the rules that have contributed more will have their fitness increases than the rules that have contributed less in this situation. If the result was a decrease in the distance then the rules that have contributed more 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 the those points in the search space that causes no improvement or degradation in the performance. The method employed in credit assignment will be discussed later.

Then the parents for the new solution are chosen proportional to their probability 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 this there is no improvement or there is degradation then the mutation is set to high probability was 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 improvement then the mutation rate is lowered proportional to this improvement until it reach a high limit (to be discussed later) , the mutation is set to zero which means that we want to keep this genetic material with no high disruption and to fine tune the solution using crossover. Binary coding is used in coding of the chromosomes.

After the GA generates a new solution the robot tries the controller with the modified rule base, if the robot had moved a certain distance with no crashing (to be determined later), then this is an ending criteria, which means that the robot had learnt this situation. Then the robot keeps this solution to the rule base and keeps it in the LTM. If not It tries from the step B skipping step C.
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 behaviors are organized in a hierarchical form and the online learning algorithm is also a hierarchy in which in the higher level we have a population of solutions stored in the LTM 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 by finding the distance it had moved before colliding. The highest fitness experience is used as a starting position to the lower level GA which is used to produce new solution to the current situation.

IV. DETAILED ALGORITHM DESCRIPTION

A. *Finding the Blamed Rules*

The robot is equipped with a Short Time Memory (STM) composed from the last 2000 actions the robot had taken using the HFLC till collision (with an obstacle or a wall). When it begins moving again using the new generated rule base the STM is initialized to record the new 2000 actions.

As mentioned before the robot starts its operation by moving using the initial rule base until collision. At the moment of collision the robot begins backing off by replaying the actions that are stored in the memory starting from the last action it had taken.

We want to determine the distance it backs off to make it escape from crashing again and to enable us to find the blamed rule for this crash. In the following analysis we will try to make all the computations related to the robot dimension, so that when we move from large robot to small robot, the algorithm can still work but with changing some parameters in the algorithm that depend on the robot dimensions. This means that our algorithm is *robot independent* and it is not developed for one kind of robot.

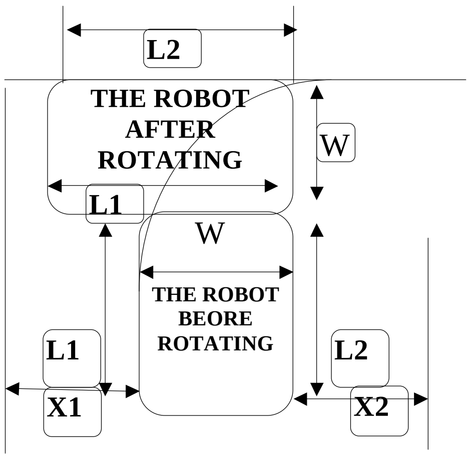


Figure (8): The Robot turning distances.

As the collision with an obstacle requires steering away from it, then we are required to find the minimum distance that if we applied maximum steering we can pass without hitting the obstacle.

The minimum front distance from which the robot if applied maximum steering can escape from hitting an obstacle is equal to the width of the robot (W) as shown in figure (8). The robot also must satisfy that at this point the left and right side will also be safe at turning. This can be satisfied be making the minimum distance from the left walls (X1) equal to L1-W and from the right side X2 equal to L2-W. Then if we made the robot back off until the minimum front sensor is equal to or just greater than W, and the minimum left sensor is equal to or just greater than X1,and the minimum right sensor is equal to or just greater than X2. In this distance if the robot applied maximum steering, 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 maneuver, in this case the robot will go back until it is possible for it to maneuver.

But doing this means that the robot at this distance 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. Also if corrected our self earlier we can avoid collision. So we will back another distance double the FB and we will call this Second Backing (SB). At the end point of SB the robot stops backing and consider this point its starting point of all the next iterations.

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 collision. So at FB we stop and find all the rules that fired at this situation and evaluate strength of each rule by how much it contributed to the final action, the greater it contributes the larger it will be blamed for collision 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.

B. *Fitness determination and Credit assignment*

In this work, we are considered in making the robot learn to adapt its combined behaviors to achieve a high level goal which is getting out of a maze while avoiding obstacles . This could be done by introducing the robot to different situations (such as corridors, obstacles, walls) and through avoiding collisions with these objects, the robot can learn these tasks by adapting its combined behaviors. In this case reinforcement is available only when the performing system collides or escapes from collision after an ending criteria. The state of the performing system where the performance is evaluated is called a reinforced state [2]. The achievement of a reinforced state may not depend only on the last action done, but also on the state from where it has been applied, i.e. , on the actions done before. This is called *delayed reinforcement* [2]. Delayed reinforcement concern tasks where the performing system needs time to express its behavior. For example; this is the case for an agent blocked in a corner that should maneuver to escape. It should apply maneuvering behavior for a given period, to be able to demonstrate its ability. Only at the end of this period it may 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 as part of the escaping maneuver. The only possibility is to evaluate the agent's performance when it succeeds in escaping, and when it is collides with that corner. From this 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 stabilizing 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 [26], the bucket-brigade algorithm may loose effectiveness as action sequences grow long, and as we use HFLC system we have long chains of rules. So we will only apply credit assignment to the rules FB and SB, as it will be shown that modifying these rules is sufficient to find a solution and there is no need to backward chaining.

The fitness of each rule at a given situation is given as follows:
By applying equation (2) and substituting C_i from equation(1) we can write the crisp output Y_t as:

$$\frac{\sum_{y=1}^{4} mm_y \dfrac{\sum_{p=1}^{M} y_p \prod_{i=1}^{G} \alpha_{Aip}}{\sum_{p=1}^{M} \prod_{i=1}^{G} \alpha_{Aip}}}{\sum_{y=1}^{4} mm_y} \qquad (2)$$

Where M is the total number of rules , y is the crisp output for each rule ,αA_i 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 behaviors.

Because we are having two output variables which are the left and the right wheel speeds, then we have Y_{t1} and Y_{t2}. Then the contribution of each rule p for a behavior y to the total output Y_{t1} is denoted by S_{r1} where S_{r1} is given by:

$$\mathbf{S_{r1}} = \frac{Y_{t1} - \dfrac{mm_y}{\sum_{y=1}^{4} mm_y} \cdot \dfrac{Y_{p1} \prod_{i=1}^{G} \alpha_{Aip}}{\prod_{i=1}^{G} \alpha_{Aip}}}{Y_{t1}} \qquad (3)$$

S_{r2} is given by:

$$\mathbf{S_{r2}} = \frac{Y_{t2} - \dfrac{mm_y}{\sum_{y=1}^{4} mm_y} \cdot \dfrac{Y_{p2} \prod_{i=1}^{G} \alpha_{Aip}}{\prod_{i=1}^{G} \alpha_{Aip}}}{Y_{t2}} \qquad (4)$$

If there is improvement of the distance, then the rules that contributed more must be given more fitness to boost their actions. If there is no improvement then the rules that contributed more must be punished by reducing their fitness w.r.t to other rules and beginning examining the solutions that were proposed the small contributing actions.
The fitness of each rule is given by:

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

where d_{new} is the distance 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. 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:

$$\mathbf{S_{rt}} = \text{Constant} - \frac{S_{r1} + S_{r2}}{2} \qquad (6)$$

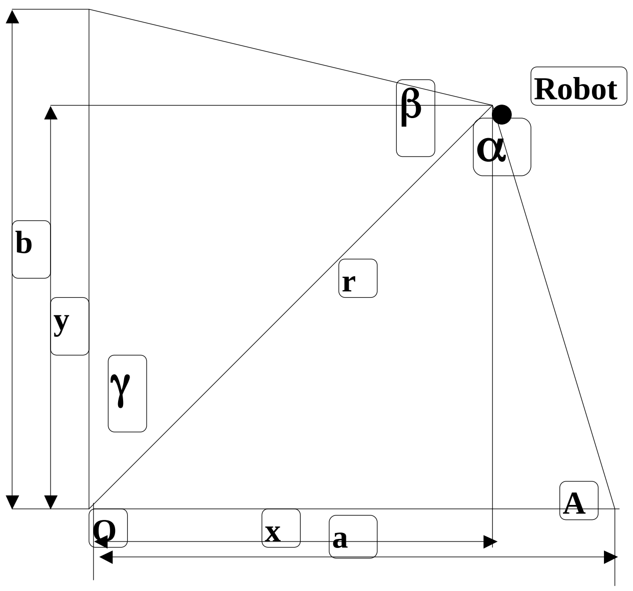


Figure (9): The Position estimation using the 3 infrared beacons.

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

1) Determination of the distance moved by the robot

In simulation, it is very easy to determine exactly, the distance the robot had moved, while in real world it is difficult.

In the lab experiments to determine this distance we have used three infrared beacons placed at right angels and at known distances **a, b,** and the infrared scanner sensor mounted on the robot gives bearing of the robot w.r.t. the three beacons. The distance of the robot from a point O (beacon number zero) is given by:

$$r = \frac{b\sin(\beta+\gamma)}{\sin(\beta)} \qquad (7)$$

where C is given by $\frac{b\sin(\alpha)}{a\sin(\beta)}$ and γ is given by:

$$\tan(\gamma) = \frac{c\sin(\beta)-\cos(\alpha)}{\sin(\alpha)-c\cos(\beta)}.$$

At the first collision of the robot (sensed by its bumper switches) and after the FB and SB. The robot at the end of SB calculates its distance r from point O which will be the original point for any new distance and is denoted by r_o. For example if the robot moves new distance r_1, then d_{new} will be equal to:

$$\sqrt{(r_1\sin(\gamma)-r_o\sin(\gamma))^2+(r_1\cos(\gamma)-r_o\cos(\gamma))^2} \qquad (8)$$

C. Long Time Memory (LTM) Application

Zhou [27] presented CSM (Classifier System with Memory) system that addresses the problem of long versus short term memory, i.e. how to use past experience 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 chunks, where every rule chunk represents a generalized version of problem solving expertise acquired in previous problem solving activity. Every time the agent is presented 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 compress the acquired knowledge into a chunk of rules that are memorized for later use in the long term memory.

In our system, as the robot begins the motion, it had no previous experience at all and the memory is empty. But as it begins learning by GA , it begins filling the memory with chunks of rules. Each rule chunk is consisting of the rules that were learnt and the actions (consequences) that were learnt by the GA.

Each time the robot is presented a situation to learn, it begins checking if the rules to be modified are present in the memory chunks or no. If for example we have rules 1,2,3,4 to be replaced and in the first chunk 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 collision then these rules are kept in the rule base of the controller and in this way we have saved the process of learning a solution to this problem from the beginning by using our memorized experience. If the robot collides again, it measures the distance it had moved to determine the fitness of the solution proposed by this memory chunks using equation (7). After all memory chunks have been examined and the robot still collides. The best solution proposed by LTM 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. This LTM will serve to speed up the search.

By doing this our system does not need the matcher calculations in [27] as our system does not use the binary message coding and the don't care conditions and always we use perfect match. We also don't need the generalizer. The chunks are laid in a queue starting from our recent experience.

The problem occurs as the system begins accumulating experience that is exceeding 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 favor this, because this means that some of the experiences the robot have discovered as solutions will be lost (which 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). So for every rule chunk 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 chunk is calculated as the product of the *frequency counter* and the *difficulty counter*, which tries to keep the rules that the robot had done a lot of effort to learn them (due to the difficulty of the situation) and also the rules that are frequently used . When there is no more room in the long-term memory, the rule chunk that had has least *degree of importance* is chosen to be replaced. If two rule chunks 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 chunk. The value of the age parameter increases over time, but is initialized whenever the associated chunk is accessed. The limit for the memory chunks is set to 2000 rule chunks, so if we exceed this limit we begin using the *degree of importance* and age operator to optimize the LTM. Of course not all the 2000 chunks will be used in each trials, because we will use the chunks that contains the rules that match the current situation.

D. Producing new solutions by Genetic Algorithms.

The GA is the rule discovery component for our system (as in the classifier system). The GA is applied to learn a new solution for a certain situation, after the solutions stored in the LTM fails. The GA produces new solutions that replaces the most two dominant rules at FB and the most two dominant rules at SB.

As mentioned earlier the GA starts by modifying the actions of the most two dominant rules at SB (to modify the earlier rules that, if their actions were true the robot can easily get out of this situation), we will call this 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, we will call this Second Replacement (SR).

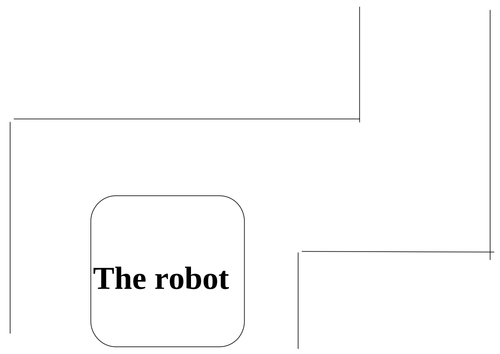


Figure (10): The robot is in a situation composed of two sub-situations, one is right turn followed by left turn.

The robot then starts moving with the modified rules. If the robot moved a distance above 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 certain 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 one situation and must be split into two situations such as figure (10). Splitting this situation into two sub-situation is essential for producing a solution.

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 great role in the GA fast convergence, in which we are interested very much. 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 value of p_c, the quicker are the new solutions introduced into the population. As p_c increases, however, solutions can be disrupted faster than selection can exploit them. The choice of 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[25].

Because we are using small population size , then we need high mutation rate to allow wider variation in the search and hence the ability to jump of the local minima. Also because we start our search of all the rules have the same consequences which means that all the genetic materials are the same, hence we need high mutation rate to introduce new genetic material with out changing the algorithm to random search. It is also desirable as the system is showing improvement in fitness (distance), the mutation rate is decreased for not loosing these genetic materials that caused this improvement and we depend on crossover to fine tune these genetic materials to obtain our 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 high mutation probability found empirically to be 0.5 (to be shown later). If there is a distance improvement we will have the mutation linearly reduced until the improvement is zero, when the improvement is equal to the robot length (determined empirically as will shown later). If the distance improvement was the same or was degradation, 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 :

$$p_m = 0.5 - \frac{0.5(d_{new} - d_{old})}{robotlength} \quad \text{if} \quad dnew > dold$$

$$p_m = 0.5 \qquad \text{otherwise} \qquad (9)$$

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 [18], so as in [18] 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 figure

(11) , in which the robot is required to learn only the right turn in a corridor. In figure (12) 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 for value of 0.1. At mutation value of 0.3 the fastest convergence was after 7 iterations with crossover value of 1.0. At mutation value of 0.5 we have the fastest convergence after 4 iterations with a crossover rate of 1.0. The same for mutation values of 0.7. At mutation values of 0.9 the system is more or less a random search and the best performance is at crossover rate 1.0 after 6 iterations. The mutation rate of 1.0 leads to no solutions at all, because this means that starting with all the genetic materials the same as our case, mutation will lead to inversion of the binary materials and we will end with all the genetic materials the same again with no new material (i.e. we will end flipping between the current genetic material and its inversion).

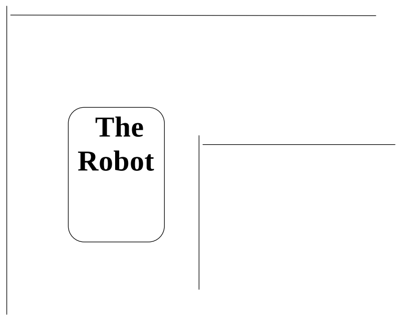


Figure 11: The problem set to the robot to learn the right turn in a corridor.

From this figure it is obvious that always as the crossover rate increases the convergence rate is faster with optimum value at crossover probability of 1.0. Also the optimum mutation value was found to be 0.5 and 0.7 but we will choose 0.5 to be our bound to decrease the risk of ending as a randomized search.

In figure (13) we conducted a series of experiments to investigate the effect of the robot length variation in equation (9) and is the robot length the optimum parameter. That is to say if we tricked the robot by saying that its size is half its original size or double, by doing this we can investigate the optimum parameter for equation (9) also we are interested in making our work robot independent (i.e. when the system is transferred to other robot it will still work). From figure(13), 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(9)).

In order to be sure that this parameter is optimum we tried varying the robot sizes and mazes size. By doing this we are 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 should the robot original length. Figures(14) shows the robot original length tried with different mazes sizes. Figure (15) shows half the robot length tried in equation(9) with different mazes sizes. Figure (16) shows third the robot size parameter in equation (8) with different robot sizes. Note that these curves stay the same as figure (13) but they converge faster as the maze size increases. But we still have the best convergence results with the original robot length in equation (9). When we substitute half the length of the robot in equation (9), a small improvement in the robot performance will falsely cause the robot to decrease the mutation so much thus causing the robot to take long time to converge, the same applies for any smaller length.

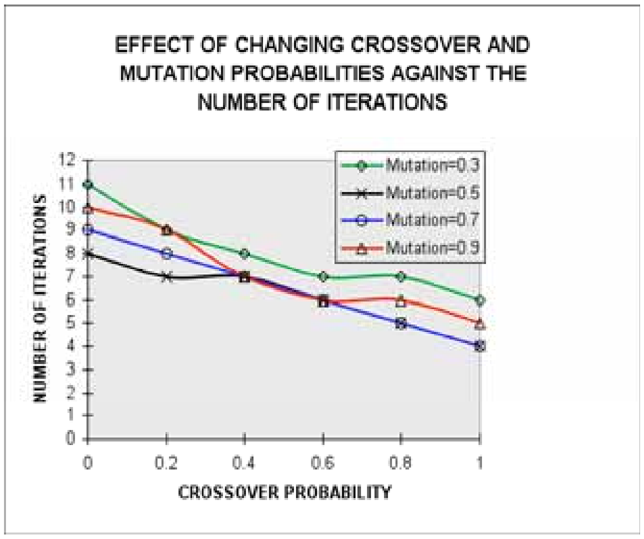


Figure 12:The convergence rate specified by the number of iterations plotted against the crossover probability for different mutation rates.

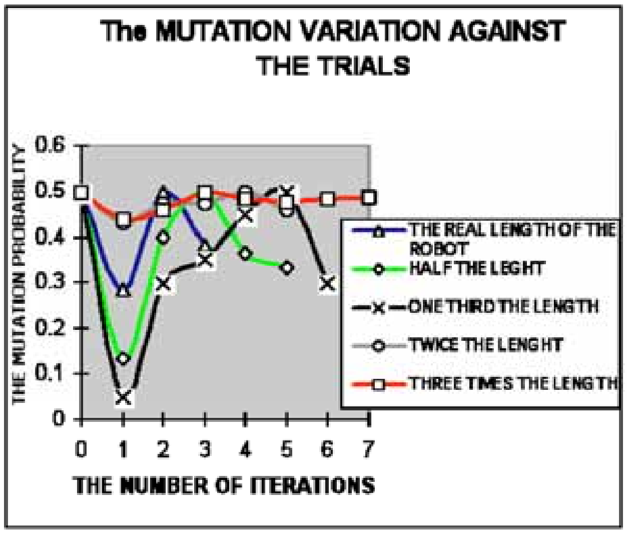


Figure 13: The effect of variation of the robot length in equation (8) over the rate of convergence (the number of iterations).

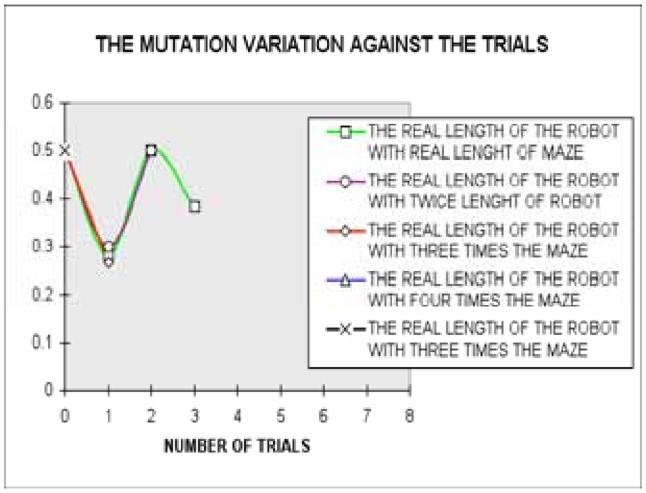


Figure 14: The robot with its original length parameter with varying maze sizes.

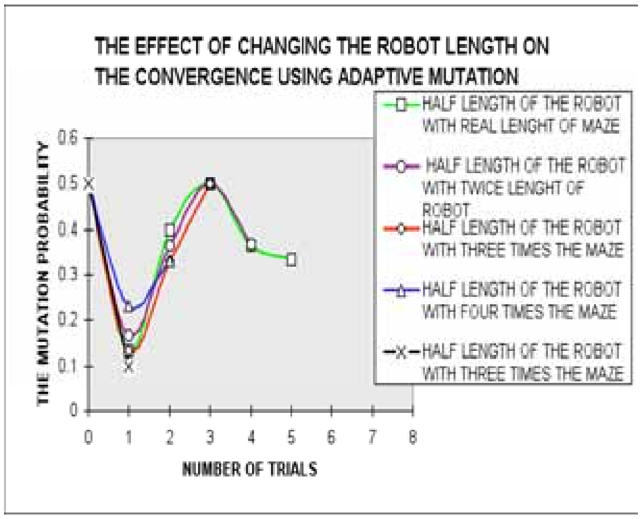


Figure 15: The robot with its half length parameter with varying maze sizes.

We use binary coding in the GA. For each rule there are two actions which are the left and right wheel velocities. As we have 4 output membership function, so we decode each action by two bits as follows, *Very Low* is 00, *Low* is 01, *Medium* is 10 , *High* is 11. So by doing this we have a chromosome length of 4 bits.

Figure (17) shows a description of the GA operation in which 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 (they have contributed more with their actions to 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 consequent of rules 1 of the obstacle avoidance and rule 2 of the right wall following which were mostly blamed at SB. The same technique is used to replace the consequents of the two dominant rules in the FB.

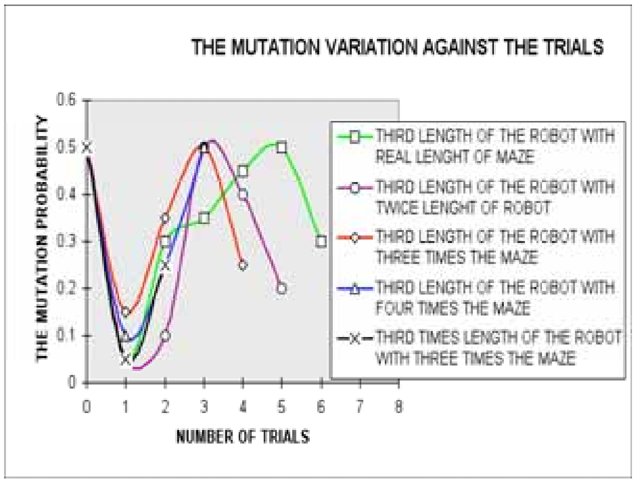


Figure 16: The robot with third its length with varying maze sizes.

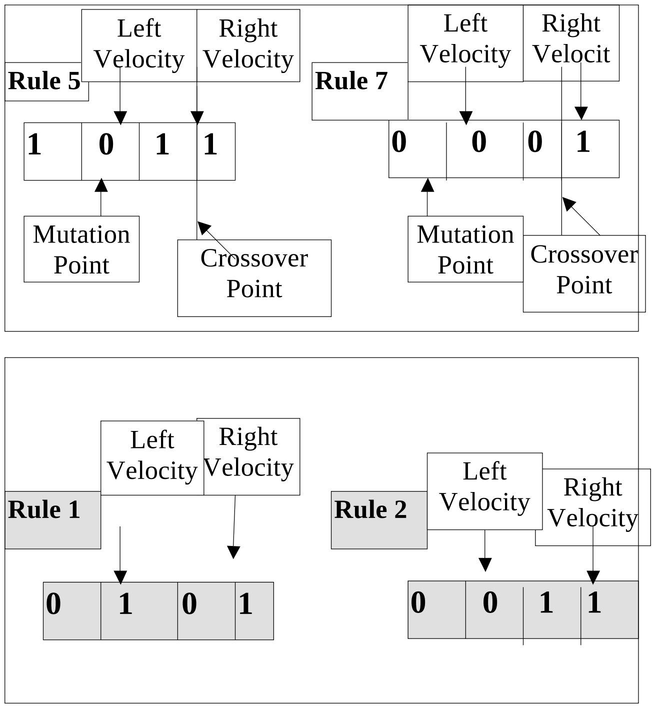


Figure 17: An example of GA processes in our proposed classifier system in which rule 5 and rule 7 (which were selected due to their higher fitness values) are generating new consequent for rules 1,2 of the FB reversal . The same will happen with two SB rules.

However to speed search we will use the ultra sound 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 sense that the left end is blocked or there is larger space to turn right. The Mechanism using the ultrasound sensors works as follows.

We will call the Left Front Sensor k1, the Right Front Sensor k2, the Left Side Front sensor k3, the Right Side Front (RSF) k5. First the robot at the SB checks if k1>k2 and k3>k5 then the direction is left , if k2>k1 and k5>k3 then the direction is right, if k3=k5 check if k1>k2 then the direction is left if k2>k1 then the direction is right. If all these condition are violated check the same for the FB to determine the direction. If every thing fails then either there is no solution and no turns can be done here and the robot must go back (which will be done any way by the robot through the backing procedure, because the robot backs off till it finds good place to start its turn) or the readings are the same because going to the left or the right is the same like going around a wall you can rotate around it from left or right so have an arbitrary direction say left.

E. The Ending Criteria of a situation:

We will try also to evaluate this criteria to be robot independent and maze independent.

We cannot use time as ending criteria because we are using variable speed, and also we cannot use the distance produced by infrared triangulation because this implies calculating the bearing while the robot is moving, and as we are using a rotating tarret to get the bearing of the beacons this implies that high imprecision in distance determination.

So we calculate the distance moved by calculating the average speed (average of the left and right speeds) over one second. Multiplying the average speed by one second should give the average distance by the robot.

As in the SB we are at minimum 2W from the front obstacle and X1 from the left and X2 from the right. If we assumed that the robot moved W without doing the right moved and at the position of FB it made the right turn , this should be rotating with a 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, the robot should escape with its sides L (maximum of L1, L2). Then the total distance moved by the robot to end a situation is given by :

$$W + \frac{\pi . W}{2} + L \qquad (10)$$

So the robot calculates the average distance moved by it every second. If this distance exceeds the distance given by equation (10), then the robot had successfully found a solution to this situation. If the number of generations exceed 6 then the distance given by (10) is reduced by half, to split this situation into two situations as described earlier.

V. Experiments and Results

The robots learns rule bases of different behaviors by learning different situations while if navigates. The robot does not learn special situations, but it learn 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. The robot learns when it needs, for example if the robot was launched in a corridor, it will learn the rules needed to navigate in this corridor and it can generalize as we will see later and navigate in different shapes of corridors, because it had learnt general and not specific rules. But when the robot is introduced to a complicated maze with left and right turn the robot must learn more rules in order to survive. This also means if the robot after learning a complete rule base, had changed its kinematics or the ground conditions is changed, the robot can still adapt itself to the environment by only adjusting small set of rules with no need to start learning from the beginning as in learning by simulation. In the first part of this section we will first introduce our system to solve difficult situations and develop a rule base that can solve other mazes easily. Next we will compare our work with some of the important work in the literature of using GA (On-line and off-line).

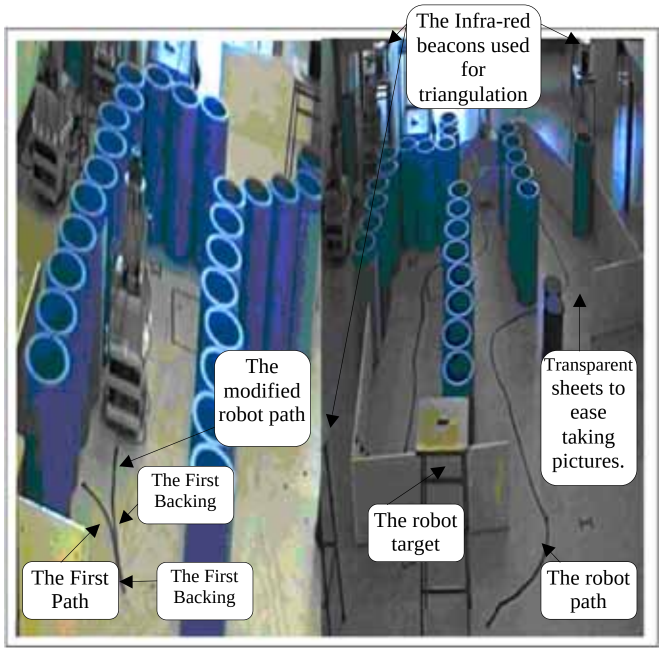


Figure (18): a)The robot learning cycle. (b) The robot path after learning.

In the following experiments we want the robot to learn the coordinated behaviors of obstacle avoidance, left, right wall following from scratch to get out of a maze without collision with obstacles. The goal seeking behavior is specified using human experience to arrive to goal after getting out of a maze. We have introduced the robot to difficult situations so that if it learnt hard situations it can learn easy situations.

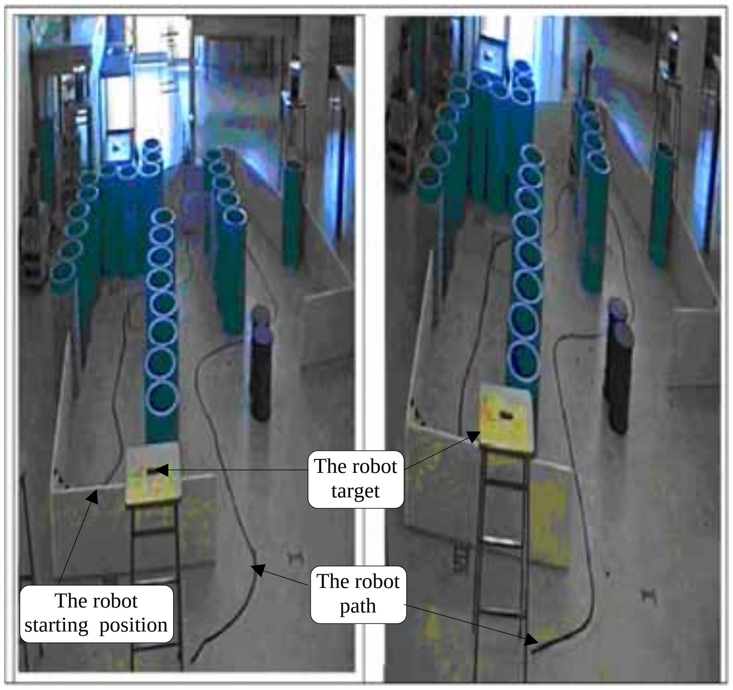


Figure (19) : a) The robot response when started from different starting position., (b) The robot response when rules are generated with different raw rules.

In these experiments we assume that the membership functions are constant and also the behavior co-ordination membership functions and rules are the constant and are set to the same values as shown above in section 2.

We have initialized the rule bases of the behaviors to be learnt 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 not sitting doing nothing.

The robot learning cycle discussed above is shown in figure (18-a) in which the robot moves then collides then it first backs (FB) and second backs (SB) and generate a modified set of rules to that situation and then it passes safely until the rules fail again and the learning cycle repeats.

The robot was first introduced to the complicated maze in figure (18-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. The robot had all its initial rule base suggesting to go forward with normal speed irrespective of any obstacles facing the robot.

After only 44 generations the robots had succeeded in getting out safely from the maze after modifying the actions of 15 rules in the obstacle avoidance behavior and 6 rules in the left wall following behavior and 6 rules in the right wall following.

Although the experiment last for about 35 minutes , most of the time elapsed concerns moving backward and forward as this takes long time due to the low speed of the robot. The computation time for each rule generation is 200 ms using 68020 20Mhz microprocessor.

After the robot gets safely out of the maze, we replace it in the starting position to test the robot repeatability and stability for 8 experiments, the robot have shown that the robot path is repeatable with in a 92 % in average and stable as it didn't crash again.

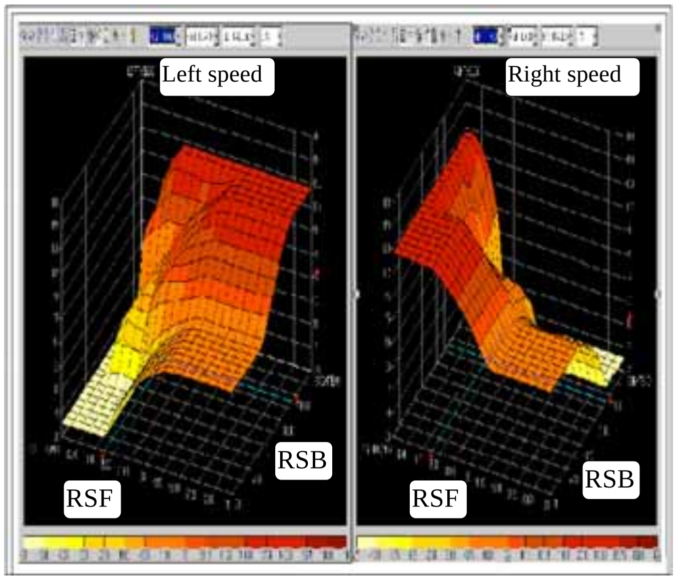


Figure (20): The control surface produced by the algorithm for the right wall following behavior, the left graph represents the RSF sensors and RSB plotted against left wheel speed, the right graph represents the RSF sensors and RSB plotted against right wheel speed.

The robot's path is shown in figure (18-b) showing it to have a smooth path through the whole maze, getting safely out towards its target. Note that the generated control surface in figure(20) is smooth and continuous

We have tried the robot at a different starting position as shown in figure (19-a) and the robot, got out safely which implies that the robot had not learnt a specific path starting from a certain point. In order to guarantee repeatability of the robot, we have started the robot with a different raw rule bases. The robot got out of the maze and found a solution after 49 generations, modifying the actions of 14 new rules in the obstacle avoidance behavior, and 6 rules in the left wall following behavior, and 6 rules in the right wall following behavior. The robot final response is shown in figure (19-b).

Although the rule bases of figure(18-b),(19-b) are slightly different, they produced a very similar response. It is difficult to determine which solution is better than the other as both solutions produce smooth control surfaces and they have a very similar response. Also almost similar rules are modified by the algorithm, which are the efficient rules in the robot motion. Thus we can assume that the solutions are almost the same. In order to be more confident in our method and be sure that the robot had learnt general rules and not a certain geometry. We have tried the robot on completely different geometry maze. This had very tight corridors and difficult turns and sparsely distributed objects. We have tried both solutions on this mazes as shown in figure(21-a,21-b). The robot in both situations showed a good response in spite of navigating tight corridors and making difficult turns. Both solutions produced similar response. In both solutions we have tried the robot with no long time memory and we have found that the robot gives the same solution after 80 iterations. This justifies the idea of LTM as besides preserving the system experience, it also speeds up the GA search as starting the GA from the best found point in the space.

The distance improvement against the number of generations (in the situation shown in figure (18-a)) is shown in figure (22), this figure show how the GA explores the space first, identifying bad regions and trying to avoid them and at the sixth iteration it succeeds in finding the solution (represented by the high improvement of 60).

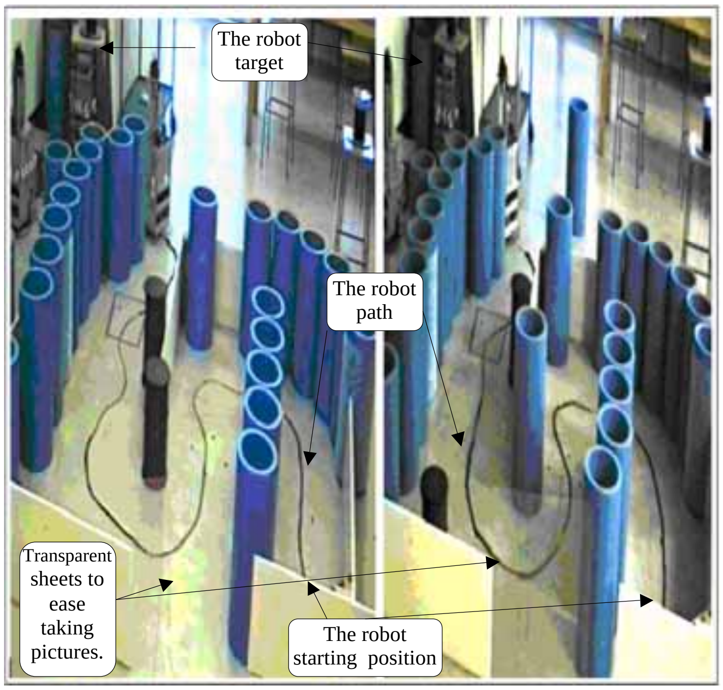

The robot target
The robot path
Transparent sheets to ease taking pictures.
The robot starting position

Figure (21):a)The robot response with the first learnt rule base tried to different geometry. b)The robot response with the second learnt rule base tried to different geometry.

In the next section, we will compare our performance with three of the most important work in the literature , Leitch [17] and Bonarini[2] and Hoffmann[12]. Leitch have used simulation in his work and Bonarini had used simulation for his robot then implemented it real robot, as hoffmann.

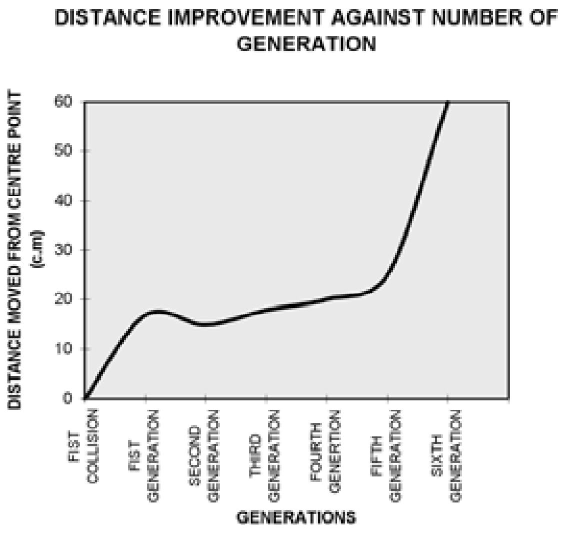
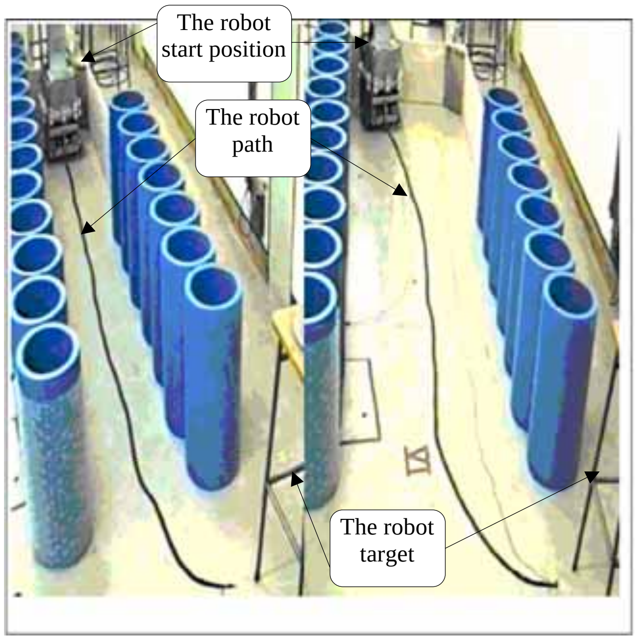


Figure (22) The Distance Improvement caused by GA to the situation of figure (18-a).

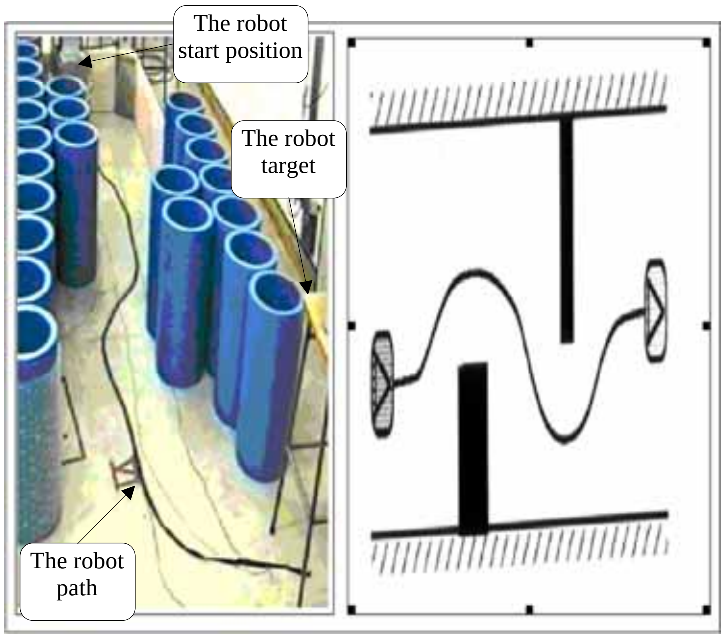


(a) (b)

Figure 23: The corridor experiments that were conducted by Leitch and Bonarini a) Tight corridor b)wide corridor.

In simulation the problem of distance determination and robot backing and robot moving speed is completely ignored because it easy to be done while in training with real robot most of the time is consumed in moving along the maze and testing the new solutions, while generating new solutions does not occupy 5% of the whole learning time. So when comparing our work with the other research we will compare with the number iterations needed to find a solution.

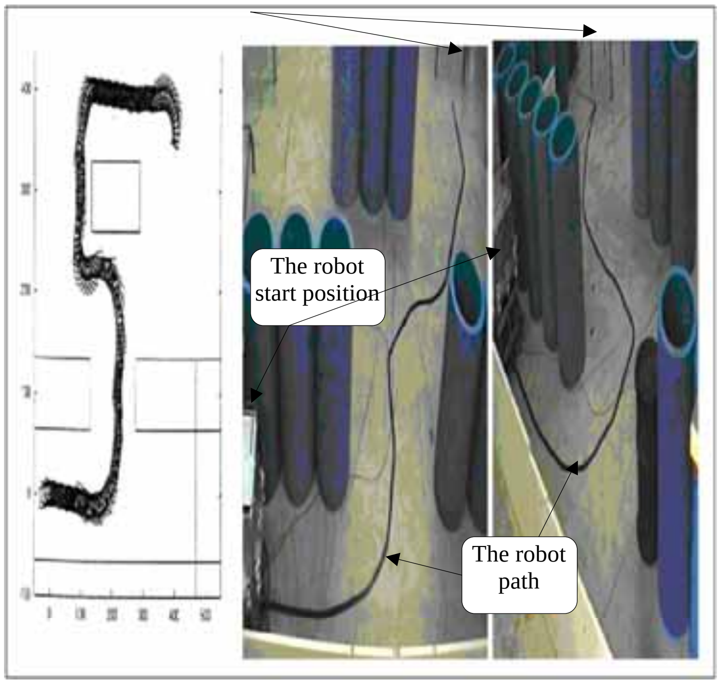
We start the comparison with [17], [2] with the conventional corridor tracking problem. We will compare the results first with [2] in which he places his 60 c.m wide robot in 3m wide corridor then he moved it to 4m and 2m wide corridors, then he placed the robot in a complicated corridor as shown in figure(24-b) . We have done our experiments with our 25 c.m wide robot preserving the same ratios with our corridors starting with 1.25 meters and then moving to 1.67m corridors and 83 cm corridors to test the portability of the rule bases[2]. But we started the learning by the hard corridor in figure (24-a) to learn most of the situations that the robot might face in a corridor. It took the robot 16 minutes (including backing time and the slow speed of the robot) to get out of the corridor and to learn the rule bases of the co ordinated behaviours. It had learnt 7 rules in the obstacle avoidance behavior, 4 rules in the left wall following behavior, 4 rules in the right wall following behavior (i.e. total of 15 rules). It have learnt these rules in an average of 20 iterations (episodes) over 4 experiments, and it follows the path shown in figure (24-a).



(a) (b)

Figure 24: comparison between our work and Bonarini's work a) Our algorithm b)Bonarini's method

The robot target



(a) (b) (c)

Figure 25: comparison between our work and Hoffmann's work a) Hoffmann's method b) Our method with target to the right c) other method with the target to the right.

The robot was started from different positions in the maze for 8 times and the robot had followed the path in figure (24-a) with a 95% degree of repeatability and with 100% degree of stability as 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 (23-a), figure (23-b) and we tested the degree of repeatability for 8 times, it was found that the degree of repeatability of the path was again 95% and the stability was 100 %. Note that the system objective function is to maximize the distance moved by the robot before crashing, and in spite of this the robot tend follow approximately the center line of the corridor, this is because every wall following behavior tries to avoid collision with its wall and because of the final balanced action, the robot follows the center line of the corridor. Leitch have tried only simple corridor following using the context depending coding and he succeeded in generating a solution after 40 generations (and his rules base should be modified again to solve the problem of figure (24-b). Bonarini have used his algorithm on a simulated robot and then he transferred this controller to the real robot. To solve the problem in figure (24-a) he needed 471 leaning episodes (iterations).

In [12] Hoffmann introduces his method of incremental tuning of fuzzy controllers by means of an evolution strategy and he gives the bench mark problem at figure (26-a), he had succeed in finding a solution after 50 iterations with a rule base of 9 rules. In his previous work he had used messy GA to learn the fuzzy controller and he had given the example in figure (25-a) (he didn't give information about the converging rate). In our algorithm we have started learning the rule bases for figure (25-a), (We didn’t learn the rule base of goal seeking as Hoffmann). After 18 minutes (including backing times and the low robot speed) the robot have achieved its goal successfully in an average of 20 iterations, learning 7 rules in the obstacle avoidance behavior and 4 rules in the left wall following behavior, and 4 rules in the right wall following behavior. The robot was tried for 8 experiments to test its repeatability and stability and it had given a path repeatability of 93% and stability of 100%. The robot is reactive as shown in figure (25-b), (25-c), as the target changes its position from the left to the right, the robot changes its path responsively following the shortest path.

The same controller was tried to the problem of figure (26-a) in which the robot moves from a tight corridor to a wide area then it finds a dead end and then it begins turning back until it is out of the whole maze. We have done the same experiment with the previous controller to test its generality. The robot successfully done the required job (following the tight corridor, finding the dead end , returning back and getting out of the corridor). This proves the generality of the learnt rules.

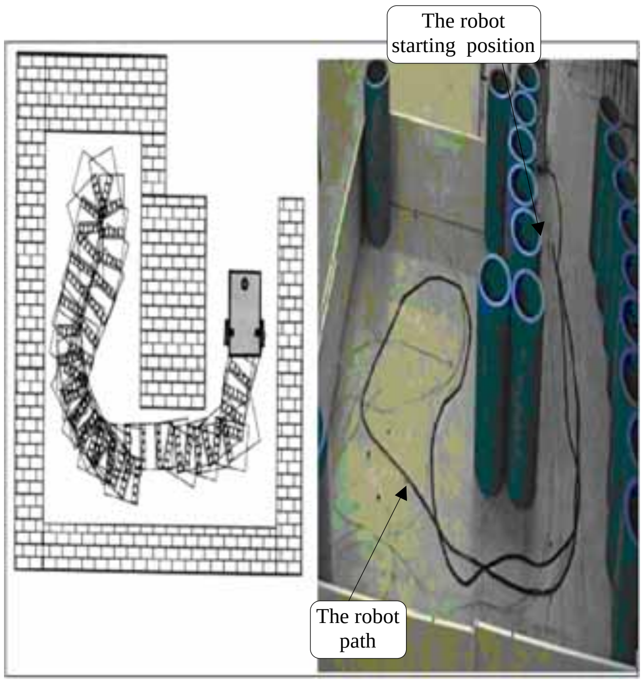


Figure 26: comparison between our work and Hoffmann’s work a) Hoffmann’s method b) Our method .

VI. Conclusions and Future Work

We have developed an on-line fast learning algorithm for learning and modifying robot behaviors from scratch. The technique uses online GA to generate the rule bases for 3 fuzzy co-operating behaviors organized in a hierarchical form. All the behaviors are learnt online with real robots and through interaction with the real world, satisfying the definition of an agent.

We have also solved the real world problems associated with learning online such as distance determination , determination of how much the robot should back when it collides with an obstacle and determination of ending condition of each situation. All these parameters were designed to be robot independent so that if the robot changes the algorithm can still work by changing only the parameters that depend on the robot size.

We have also developed a long time memory technique in which the robot memorizes all its previous solutions so that it can use them when faced by similar situations in the future, this aids the robots to find solutions with out even needing GA learning and if the robot still crashes it selects the most appropriate solution to this solution to serve as a starting point to GA which was shown to reduce the learning time.

The proposed system can be viewed as a double hierarchy system in which the fuzzy behaviors are organized in a hierarchical form and the online learning algorithm is also a hierarchy in which in the higher level we have a population of solutions stored in the LTM 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 by finding the distance it had moved before failing. The highest fitness experience is used as a starting position to the lower level GA which is used to produce new solution to the current situation.

This technique is adequate for outdoor robots where the dynamics of the robot as well as the environment is rapidly changing and requires the robot to quickly modify itself to these changes .

The algorithm is very fast in finding an appropriate solution; it finds a solution fast compared even with simulation techniques and the methods found in the literature for fuzzy robot controller design using GA.

In order to test the generality of our technique we have experimented with starting from different points and using different initial populations, we have also used completely different geometrical mazes. The robots have shown a constant response and smooth response to all these changes which suggest that it had learnt general and not specific rules.

The learnt rules will not be frozen as they can be modified when they fail a certain situation. In this event the whole behavior will not be learnt, as only the part of the rule base that has done badly.

For the future work we will try to learn how to coordinate these behaviors together. We will also try to learn the membership functions of the different behaviors and apply these technique to outdoor robots.

REFRENCES

[1] J. Albus, “Mechanisms of planning and problem solving in the brain”, Mathematical biosciences, pp 247-295, 1979.

[2] A. Bonarini F. Basso, “ Learning Behaviours implemented as fuzzy logic and reinforcement learning”, second online workshop on evolutionary computation ,1996.

[3] P. Bonelli, “ A new Approach to fuzzy classifier systems. Proceedings of the fifth International conference on genetic algorithms , pp223-230, 1993.

[4] R. Brooks, “ Artificial life and real robots “, MIT press , 1992.

[5] M. Dorigo, M. Colombetti, “ Robot Shaping : Developing situated agents through learning”, Artificial Intelligence Journal , 1993

[6] M. Dorigo, “ Genetics-based machine learning and behaviour based robotics: A new synthesis”, IEEE transactions on system, man and cybernetics, pp 141-154, 1993.

[7] T. Furuhashi, O. Nakaoka” Controlling excessive fuzziness in a fuzzy classifier system”, Proceedings of the Fifth International conference on genetic algorithms, pp. 635-637, 1993.

[8] D. Goldberg , “ Genetic Algorithms in search, optimization and machine learning”, Addison-Wesley, Reading , MA , 1989.

[9] S. Goodridge, M. Kay, R.Luo, “ Multi-layered fuzzy behaviour fusion for reactive control of an autonomous mobile robot”, Fuzz-IEEE 1997, pp. 579-584, 1997.

[10] H. Hagras, V. Callaghan, M. Colley, M. West, “ A behavior based hierarchical fuzzy control architecture for agricultural autonomous robots”, The International Conference on computational intelligence for modelling, control and automation, 1999.

[11] F. Herrera, O. Cordon, “GA and Fuzzy logic in control processes”, technical report # DECSAI-95109, University of Granada, March 1995.

[12] F. Hoffmann , “Incremental tuning of fuzzy controllers by means of evolution strategy”, GP-98 Conference, Madison, Wisconsin, 1998.

[13] J. Holland , “Genetic Algorithms and classifier systems : foundations and future directions”, The MIT Press, Cambridge , Massachusetts, The MIT Press edition ,1992.

[14] C.Karr , “ Applying Genetic algorithms to fuzzy logic”, AI Expert , March 1991, pp38-43.

[15] C. Lee, “ Fuzzy logic in control systems: Fuzzy logic controller, PartI, II, “, IEEE Trans on syst, Man, Cybern, Vol.20 , pp. 404-432, 1990.

[16] C. Lee, Takagi , “Embedding Apriori knowledge into an integrated fuzzy system design methodbased on genetic algorithm”, Proc of the Fifth IFSA World Congress, pp1293-1296, 1993.

[17] D. Leitch, “A new genetic algorithm for the evolution of fuzzy systems, PhD thesis, University of Oxford, 1995.

[18] G. Linkens, O. Nyongeso, “GA for fuzzy control, part2: Online system development and application “, IEE proc control theory appl, Vol. 142, pp. 177-185, 1995.

[19] O. Miglino , H. Lund, S. Nolfi., “ Evolving Mobile Robots in Simulated and Real Enviroments. Artificial Life” Technical Report NSAL-95007, Reparto di Sistemi Neurali e Vita Artificiale, Istituto di Psicologia, Consiglio Nazionale delle Ricerche, Roma.

[20] M. Minsky , “Berechung: Endliche and unendliche Maschinen. Verlag Berliner Union GmbH, Stuttgart.

[21] J. Qin , M. Walters, “A GA-based learning algorithm for the learning of fuzzy behaviour of a mobile robot reactive control system”, GA in Engineering systems, Innovation and applications, pp 251-258 , 1997.

[22] P. Reigniev, “ Fuzzy logic techniques for mobile robot obstacle avoidance”, Robotics and Autonomous system, 12/3-4, pp 143-153, 1996.

[23] V. Rendon ,. “ A classifier system for continously varying variables”, Proc of the Fourth Int. Conf. On genetic algorithms ,pp. 346-353, 1991.

[24] A. Saffiotti ; Fuzzy Logic in Autonomous Robotics : behavior coordination.; proceedings of the sixth IEEE Int Conference on Fuzzy systems , Vol .1 pp 573-578, 1997.

[25] M. Srinivas, L. Patnaik, “ Adaptation in genetic algorithms”, Ga for pattern recognition, pp. 45-64, 1996.

[26] S. Wilson, “ Hierarchical credit allocation in a classifier system “, In genetic algorithms and simulated anealing, pp.104-105, 1987.

[27] H. Zhou, “ A computational model of cumulative learning”, Machine learning journal, pp. 383-406, 1990.