Obstacle Avoidance Behaviors for Mobile Robots Using Genetic Algorithms and Recurrent Neural Networks

Jesus Savage, Stalin Muñoz, Mauricio Matamoros, Roman Osorio

Bio-Robotics Laboratory, School of Engineering, Universidad Nacional Autónoma de México

Abstract:
This paper discusses how to generate mobile robots' behaviors using genetic algorithms (GA). The behaviors are built using state machines implemented in recurrent neural networks (RNN), controlling the movements of a humanoid mobile robot. The weights of the RNN are found using a GA, these are evaluated according to a fitness function that grades their performance. Basically, this function evaluates the robot's performance when it goes from an origin to a destination, and the grading of the robot evaluates also that the robot's behavior using RNN is similar to the behavior generated by a potential fields approach for navigation. Our objective was to prove that GA is a good option as a method for finding behaviors for mobile robots' navigation and also that these behaviors can be implemented using RNN.

Keywords: State Machines using RNN; Mobile Robots Behaviors; Genetic Algorithms

1. INTRODUCTION

In mobile robots, behaviors are used for navigation [1], and these methods can be implemented using state machines.

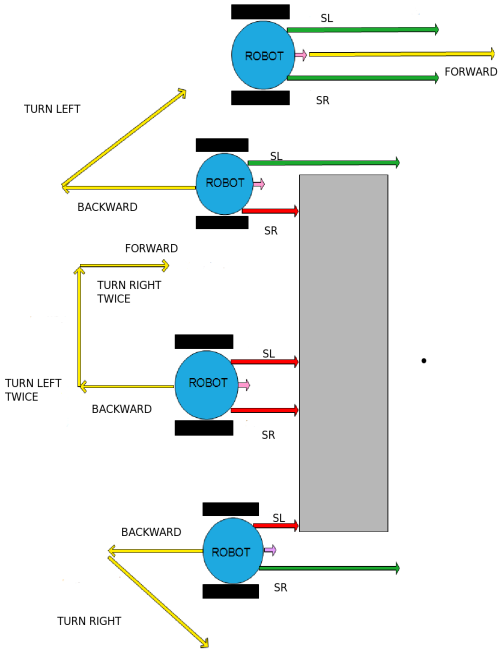

Fig. 1. Robot avoiding an obstacle

Savage [2] proposed a FPGA-based behavior implementation for a mobile robot that avoids obstacles. Figure 1 shows this behavior, the robot has two sensors in its left and right side, that allows it to detect obstacles. Figure 2 shows the algorithm state machine (ASM) for the obstacle avoidance behavior.

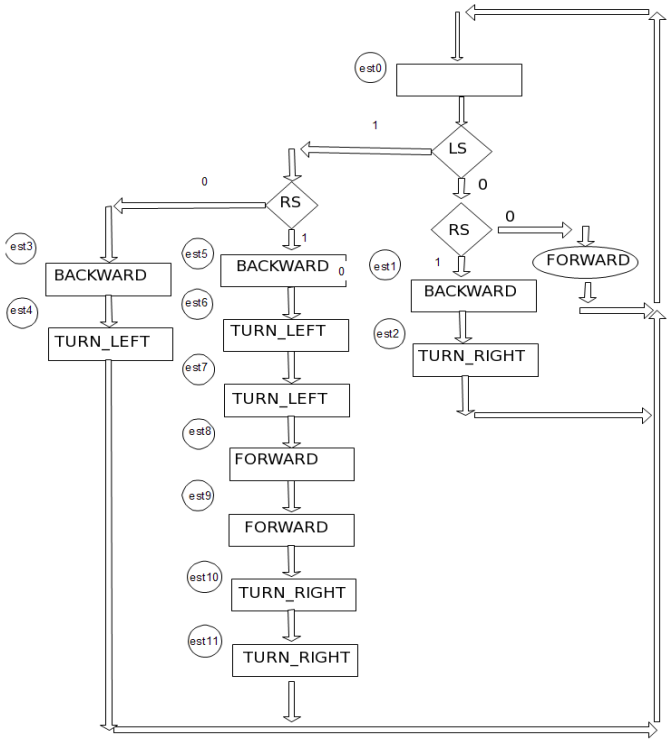

Fig. 2. A simple algorithm state machine for a mobile robot that avoids obstacles

[1] This work was supported by PAPIIT-DGAPA UNAM under Grant IN-107609

There is a lot of interest in the robotics community on how to generate this kind of algorithms automatically, specifically how to generate algorithm state machines and its implementation. There are several methods to build state machines, in this paper is described one where RNN are used. Ziemke [3] proposed the use of RNN to create the behavior of a mobile robot that takes objects.

In our research we were interested in how to create reactive behaviors for mobile robots to avoid unknown obstacles, specifically we were interested in how to emulate an insect like behavior of a robot that uses a potential fields approach for navigation using RNN. The RNN is trained in such way that problems associated with the potential fields approach, like getting stuck in a local minimum, are solved.

In the following sections are explained each of the components that intervene in our approach: potential fields, RNN, GA, the results from the experiments performed and finally conclusions.

2. BACKGROUND

2.1 Potential Fields

Under this idea the robot is considered as a particle that is under the influence of an artificial potential field U whose local variations reflects the free space structure that it depends on the obstacles and the destination goal that the robot needs to reach [4].

The potential field function is defined as the sum of an attraction field that pushes the robot to the goal and a repulsive field that takes it away from the obstacles. The robot's movements are done by iterations, in which an artificial force is induced by:

$$F(q) = -\nabla U(q) \tag{1}$$

that forces the robot to move to the direction where the potential field decrees, where ∇ is the gradient in q and $q = (x, y)$ represents the coordinates of the robot position.

The potential field is generated by and the repulsive field U_{rep}

$$U(q) = U_{atr}(q) + U_{rep}(q)$$

thus

$$F(q) = F_{atr}(q) + F_{rep}(q)$$

where

$$F_{atr}(q) = -\nabla U_{atr}(q)$$
$$F_{rep}(q) = -\nabla U_{rep}(q)$$

Repulsive potential field The goal of the repulsive potential field is to create a force that takes away the robot from the obstacles, this is obtained using a potential value that tends to infinite in the surface of the obstacle and decreases as the robot goes away from it. The following equation shows a field with the previous characteristics

$$U_{rep}(q) = \frac{1}{2}\eta \frac{1}{\|q - q_{obstacle}\|^2} \tag{2}$$

where $q_{obstacle}$ represents the coordinates of the obstacle. For several obstacles, the total field potential is the superposition of the individual potential field of each obstacle,

$$U_{rep}(q) = \sum_{i=1}^{k} U_{rep}^{k}(q) \qquad \text{where } k = \text{obstacle number}$$

Attractive potential field Attractive field potential creates an attraction force through the goal configuration of the robot. It can be considered a parabolic field of the following form

$$U_{atr}(q) = \frac{1}{2}\varepsilon_1 \|q - q_{destination}\|^2$$

where $q_{destination}$ represents the coordinates of the destination.

There are several methods for planning using potential fields, one of them is to use the gradient vector to guide the robot from the initial position to the goal. Defining the Unitarian vector pointing to the gradient direction

$$f(q) = \frac{F(q)}{\|F(q)\|}$$

in this way the movement in discrete times is defined by

$$q_{i+1} = q_i + \delta_i f(q), \tag{3}$$

where δ_i is an step constant.

One of the main problems using this technique for planning the robot's movements is that the planner can stuck in a local minimum where the attraction force and the repulsion forces cancel each to other, thus the movement of the Robot is zero or it oscillates around a path.

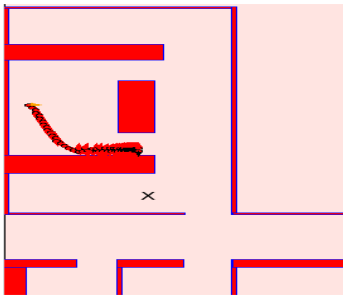

Fig. 3. Robot got stuck in a local minimum

Figure 3 shows a top view of an environment, where the obstacles are represented by red polygons and the robot by a circle, when the robot found an obstacle in the middle of the path between the origin and the destination goal, represented as an X in this figure, it got stuck in it, oscillating back and forth, due to the repulsion and attraction forces. First the repulsion forces repealed the robot from the obstacle, and when the robot is a little far away from it the attraction force pushed it back to the obstacle, and then the repulsion force acts again repeating the whole process.

One method to eliminate this by adding additional attraction forces in the space. Figure 4 shows a topological map of the environment, then given an origin and destination, a search algorithm, Dijkstra, finds a set of nodes

$N = (n_1, n_2, ..., n_k)$ that the robot needs to reach using the potential fields navigation approach. After reaching node n_i the next node is n_{i+1} and so on until it reaches node n_k, using this approach the robot reach its destination as it is shown in the same figure.

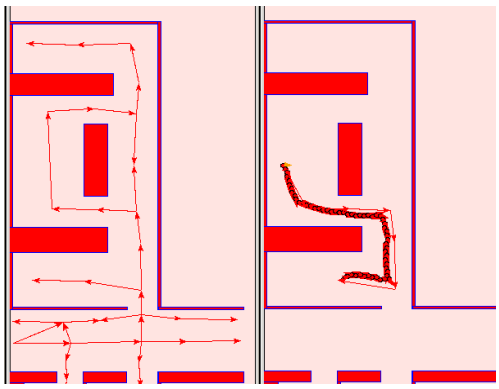

Fig. 4. The figure's left side shows the topological map of the environment, the right side shows the path generated by a potential fields approach combined with the Dijkstra algorithm

2.2 Recurrent Neural Networks

A RNN is a type of neural network in which some of the outputs are feedback as inputs with a delay, see Figure 5, these feedback neurons are the ones that keep the state variables. The output of each neuron in the network is:

$$y_i^l = f(\sum_{j=0}^{N_{l-1}} w_{ji}^l x_j^l)$$

where l represents the layer; i the neuron in layer l; w_{ji}^l represents the neurons' weights; N_{l-1} represents the number of neurons in layer $l-1$; x_j^l represents an input coming from the output of neuron j in layer $l-1$, in the first layer the inputs come from the sensors; finally $f()$ is a non linear function.

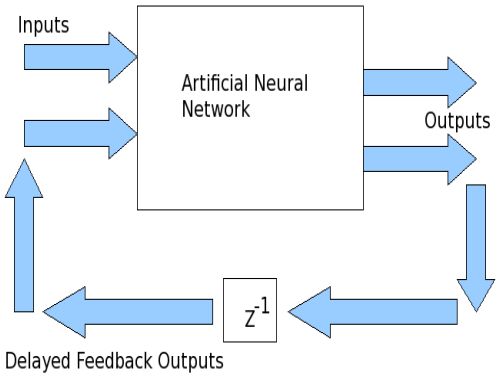

Fig. 5. Recurrent Neural Network

2.3 Genetic algorithms

One of the objectives of our research was how to generate the algorithms of state machines for mobile robots automatically, Wadhe [7] showed that these type of algorithms can be found by GA techniques.

GA are part of evolutionary computing, they are based on Darwin's Evolution theory. Systems that use this learning method improve their performance through a process that simulates reproduction by the surviving of the fittest individuals. First, a population of individuals is created, represented by a set of chromosomes, which consist of genes that can have specific values called alleles; the set of chromosomes is named genome. The population has the capacity for reproduction, recombination (crossover) and mutation in order to create offspring, where the individuals better adapted are selected to form the new individuals' generation. All the terminology used is related to biological evolution [8].

Selection Individuals from the old generation are selected to be the parents of the new offspring. There are several methods to select the individuals, like a roulette wheel; stochastic universal; elitist; Vasconcelos and others. All these methods lead to better individuals in the new population, or at least to preserve the better parents, in our approach we used the Vasconcelos selection, that was developed by Kuri et al [9].

Crossover In crossover two individuals are considered the parents, and one section of each of their chromosomes are selected, randomly, and they are interchanged, thus two off-springs are generated.

Mutation When there is mutation an individual could change, randomly, one section of its chromosome, creating a new individual. Frequently, the mutation is used to get out the genetic algorithms from local minimums.

Fitness Each individual belonging to a population is evaluated, and it is assigned a fitness value according to its performance. Fitness is very important for the selection of the best individuals from a population to generate a new one.

3. *GENETIC ALGORITHM FOR FINDING NAVIGATION BEHAVIORS*

A GA is used to automatically generate a navigation behavior, algorithm state machine, that would allow a mobile robot to navigate in an environment avoiding obstacles.The following GA finds the RNN weights, $B = \{w_{11}^1, ..., w_{1I}^1, ..., w_{11}^m, ..., w_{1J}^m, ..., w_{11}^n, ..., w_{1K}^n\}$ that form the behavior:

1. First a population is generated randomly, with L individuals $B_1, B_2, ..., B_L$, in which each individual's chromosome is a string of binary numbers that represents the behaviors $B_i = \{011011...0101\}$.

The string is separated into small segments that represents the neural network weights, $\{w_{ij}^k\}$, that is coded using a binary number $\{(0011010...011)\}$ and using N bits for the integer part and the rest for the fractions.

2. Each individual (chromosome) is evaluated giving a fitness value according to the individual's performance. The simulated mobile robots have K discrete times to go from an origin to a destination. The fitness function evaluates the distance between the last position reached

and the goal, the number of times the robot hits an obstacle, the number of steps used to reach the goal and also the number of times the robot went backward.

Then the fitness function is:

$$fitness(i) = K_1/(DistDest_i + 1.0) + K_2/nSteps_i + K_3/(nStuck_i + 1.0) + K_4/(nBack_i + 1.0) + K_5 * Path_i$$

Where $DistDest_i$ is the distance between the last position of the robot and the destination; $nSteps_i$ has the number of steps used to reach a goal; $nStuck_i$ has the number of times the robot hits an obstacle; $nBack_i$ has the number of times the robot went backward; $Path_i$ evaluates how close is the path that the robot followed compared to the one generated by a potential fields navigation approach, using a topological map and the Dijkstra algorithm (PFTDA). Figure 6 shows the topological map of the environment, a path generated by the potential fields navigation using as attraction points the nodes found by the Dijkstra algorithm and a path found using a RNN. In this case the individual i that represents the RNN obtains a high value for $Path_i$ because both paths are very similar.

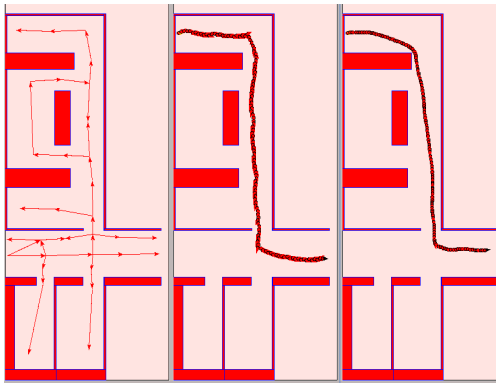

Fig. 6. The figure's left side shows the topological map of the environment, the middle side shows the path generated by a potential fields combined with the Dijkstra algorithm and the right side shows the path generated using RNN

Constants K_1, K_2, K_3, K_4 and K_5 are used to tune the performance of the robot according to its specifications, for instance, if it is desired that the final robot's behavior goes backward only few times during navigation, then K_4 should be several times bigger compared to the other constants.

3. Select the best individuals according to their fitness function and create a new population with individuals generated trough evolutionary's operators (selection, crossover and mutation).

The selection is done using the Vasconcelos GA, this algorithm was chosen due to its good performance in our previous research involving mobile robots [2] , in it the individuals are first ranked, according to their performance, from the best one to the worst one, then new individuals are generated by combining them.

4. The offspring and their selected parents form the new population.

5. Iteration from 2 to 4 is repeated for N generations or until the overall fitness criteria between two generation is less than a given ϵ.

3.1 Simulator

For each generation the GA needs to evaluate population's individuals, doing this with the real robot it would require too much time, that would be impossible to do. Thus, the GA needs a simulator, as close as it can be to the real robot and its environment. The simulator gets the individuals chromosomes and executes the algorithm state machine represented by them, it simulates the movements of the robot depending of the output generated in the present state and the simulated robot's sensors.

To simulate the sensors readings, the objects in the environment are represented by a set of lines, the sensors' values are obtained by the intersection between a line coming from each of the sensors and a line that belongs to an object [10]. The sensor value is then the distance between the sensor position and the position of the intersection with a line of the obstacle.

The simulated mobile robot has an array of laser sensors, in figure 7 the rectangle represents an obstacle, the circle represents the robot and the lines represent the sensors readings.

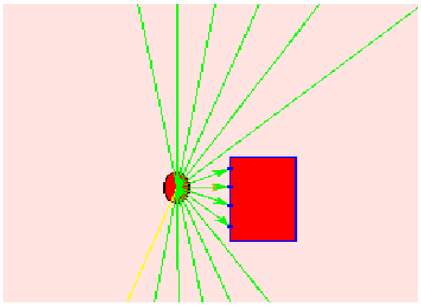

Fig. 7. Simulation of the robot's proximity sensors

The simulator can add Gaussian or uniform noise to the sensors values, as well as to the robot's movements.

4. *EXPERIMENTS AND RESULTS*

The system was tested with the simulator first with a simple environment shown in figure 7, and later with a more complex environments with rooms and corridors shown in figure 6. In figure 7 the red rectangle represents an obstacle of 1.0 x 0.5 m. and the diameter of the robot is 0.3 m. The inputs of the RNN are divided in three groups: laser readings, attraction vector to the destination and the feedback outputs, that represent the states of the state machine, as it is shown in figure 8.

The laser sensor returns 682 values that are grouped together to form the inputs for the RNN

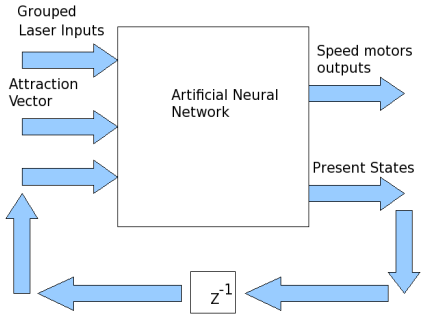

Fig. 8. Robot's State Machine Using a Recurrent Neural Network

$$S_a(i) = \frac{1}{Nr_i} \sum_{j=R_i}^{R_i+Nr_i} s_j$$

where Nr_i = is the number of sensors' values in region i, and R_i is the initial sensor index for the same region.

For training, first the GA obtained individuals capable to follow straight lines. The size of the population was 100 individuals, with 0.7 of probability of crossover, 0.01 of mutation rate, with 100 generations and using the Vasconcelos GA. The system was tested with different RNN configurations: number of laser groupings, number of hidden layers, number of neurons per layer, number of recurrent neurons and number of bits used for the weights.

Figure 9 shows the fitness function of the best individual and population average for the weights of a RNN with 4 inputs; 1 hidden layer with 10 neurons; 4 outputs, 2 for controlling speeds wheels and 2 for representing the states of the recurrent network; with 100 individuals; using the Vasconcelos Algorithm with 50 generations.

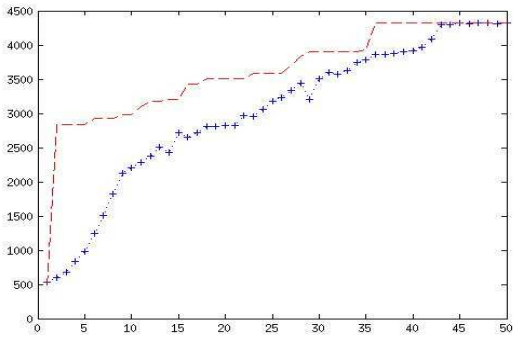

Fig. 9. Fitness function of the best individual, -, and population average, +, with 4 laser groups

Figure 10 shows the fitness function with the same conditions but with 8 laser inputs, figure 11 with 16 and figure 12 with 32.

As we can see from these figures with 8 and 32 laser groupings the populations converges with less generations, but even tough for 16 laser groupings needs more generations to converge it has the best performance.

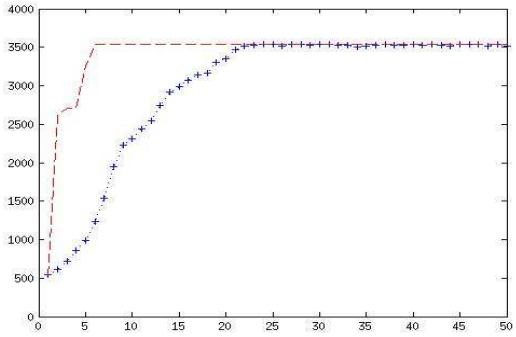

Fig. 10. Fitness function of the best individual, -, and population average,+, with 8 laser groups

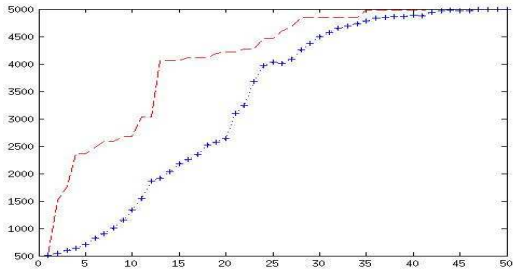

Fig. 11. Fitness function of the best individual, -, and population average, +, with 16 laser groups

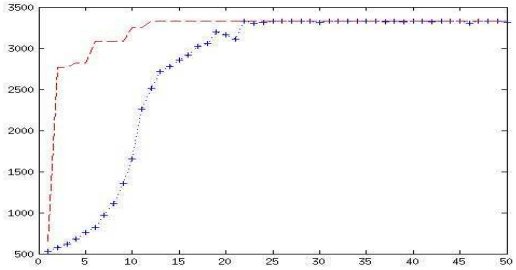

Fig. 12. Fitness function of the best individual, -, and population average,+, with 32 laser groups

After it was found a set of individuals that go in straight line, the GA obtained individuals able to evade the obstacle, for this a topological map that surrounds the obstacle was used during training with the potential field approach as a mentor.

Figure 13 shows the fitness function of a population that knew how to go in straight lines and it was retrain to avoid obstacles.

Figure 14 shows the path found by the PFTDA used to train the RNN and figure 15 shows the best population's individual that mimics that path.

The best individual obtained was tested 100 times, in the simulator, to avoid this obstacle and 95% of the times it reached the destination.

The system was tested with more complex environments with rooms and corridors, shown in figure 6. In this type of complex environments the simulated robot with the

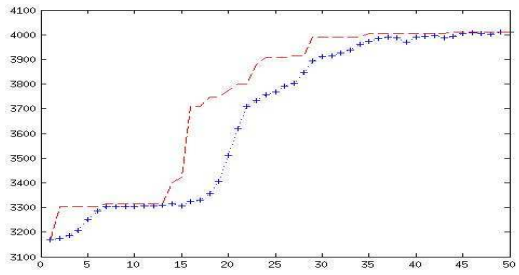

Fig. 13. Straight path fitness individual, -, and population average,+, with 32 laser groups

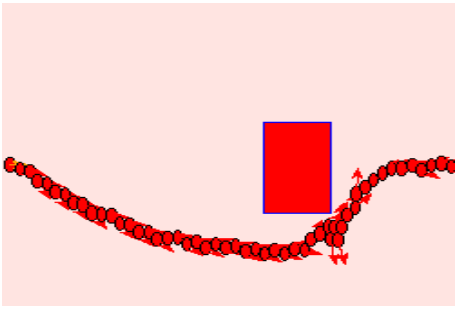

Fig. 14. Path found by the potential fields algorithm

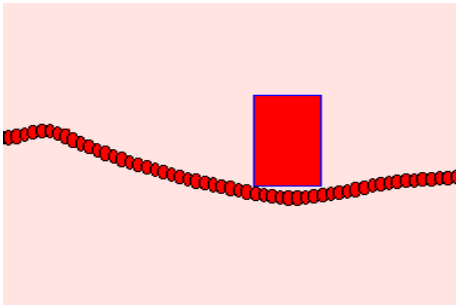

Fig. 15. Path found by the RNN with 32 laser groupings that avoids an obstacle

learned state machine behavior implemented using a RNN was able to learn how to leave rooms, follow corridors and follow complex paths.

5. CONCLUSIONS

In this paper we presented how to generate mobile robots' behaviors using GA. An obstacle avoidance behavior was found automatically using GA and it was successfully implemented in a RNN. We proved that GA is a good method for finding the weights of RNN, that mimics a potential field navigation approach without having the problem of getting stuck in a local minimum and without using a topological map. One of the advantages of using a RNN is that the machine's states are represented with continuous values instead of using discrete ones.

In future work the best RNN behavior obtained will be tested in the humanoid robot shown in figure 16, that uses a Kinect device for detecting the environments.

Fig. 16. Humanoid robot Justina

REFERENCES

[1] Arkin, R.C. Behavior-Based Robotics- MIT Press, Cambridge, MA, 1998.

[2] Jesus Savage, Marco Morales. Laboratory Assignments to Teach the Basics of Programmable Logic Applied to Mobile Robots, Primer Taller de Computo Reconfigurable y sus Aplicaciones en Educacion e Ingenieria, 2010, Cancun, Quintana Roo.

[3] Ziemke, T, Remembering How to Behave: Recurrent Neural Networks for Adaptive Robot Behavior. Recurrent Neural Networks, CRC Press, 1999, 355-384., US.

[4] J.-C. Latombe, *Robot Motion Planning*. Massachussets, USA: Kluwer Academic Publishers, 1991.

[5] Brooks, R. A. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, RA.-2(1):14-23, 1986.

[6] Sunggu Lee. Advanced Digital Logic: State Machine Design Using VHDL, Verilog, and Synthesis for FPGAS, 2005.

[7] Mattias Wahde, *An Introduction to Adaptive Algorithms and Intelligent Machines*. Chalmers University of Technology, Goteborg, Sweden, 2002.

[8] Dario Floreano and Claudio Mattiussi. Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies by Dario Floreano and Claudio Mattiussi, MIT Press, MA, 2008.

[9] Kuri-Morales, A., "The Application of Genetic Algorithms to the Evaluation of Software Reliability", Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques, IGI Global, pp. 100-120, Editor(s)): Monica Chis, ISBN: 9781615208098, 2010

[10] J.-C. Latombe, *Robot Motion Planning*, Massachussets, USA: Kluwer Academic Publishers, 1991.

[11] Terasic Technologies. Max II Micro UserManual release v1.32, 2009.