# Mobile Robots' Behaviors Derived with Genetic Algorithms and Implemented Using FPGAs

Jesus Savage, Roman Osorio, Marco Negrete, Mauricio Matamoros, and Jesus Cruz

BioRobotics Laboratory, School of Engineering, Universidad Nacional Autónoma de México

***Abstract*—This paper presents how to generate mobile robots' behaviors, using state machines, and implemented with FPGAs. These behaviors are automatically generated by a genetic algorithm, GA, and are encoded in such a way that a state machine architecture executes them, controlling the overall operation of a small mobile robot. The behaviors generated by the GA 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 it grades that the robot's behavior using a state machine, built with FPGAs, is similar to the behavior generated by a potential fields approach for navigation. In our approach each individuals' chromosome represents, given a set of inputs coming from the sensors and the current state, the next state and outputs that controls the robot's movements. Our objective was to prove that FPGAs are a good option for the implementation of mobile robots' behaviors, due to the parallelism that can be achieved with them, and also that these can be automatically found using GA.**

***Keywords*-FPGAs; State Machines; Mobile Robots Behaviors; Genetic Algorithms.**

## I. Introduction

In mobile robots, behaviors are used for navigation and these methods can be implemented using state machines. Rodney Brooks proposed a robotics paradigm to control robots [1], in which their behaviors are build using augmented state machines (AFSM), see figure 1.

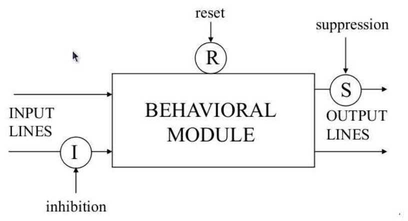

Figure 1. Augmented Finite State Machine

In an AFSM some of its inputs and outputs can be substituted for other values externally by another AFSM. Then, by connecting together the AFSMs, each one containing a behavior, emergent intelligence can be achieved by a robot. In the subsumption architecture, the inputs and outputs of each of the robot's behaviors, AFSMs, can be canceled by other AFSMs depending of their hierarchy in the system, figure 2 shows one example of this architecture.

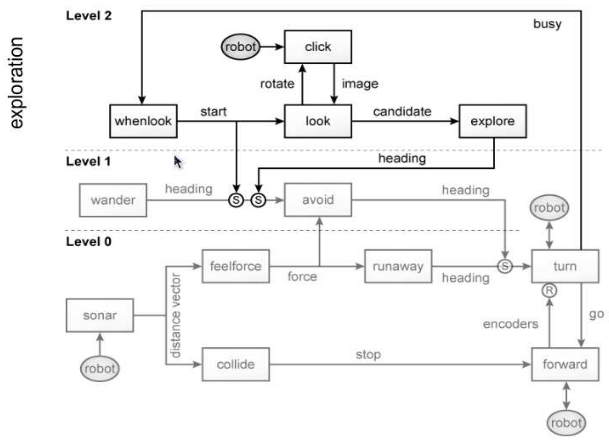

Figure 2. An example of Brooks' subsumtion system that controls a robot [3]

FPGAs are a very good option to implement subsumption architectures due to the parallelism that can be obtained with them, where several AFSMs can be working at the same time, having a better performance if the same architecture would be implemented using a procedural language in micro-controller.

Savage [2] proposed a FPGA-based behavior implementation for a mobile robot that avoids obstacles. Figure 3 shows this behavior, the robot has two sensors in its left and right side, that allows it to detect obstacles. The robot moves through two motors: it translates when the motors moves in the same direction, and it turns on its axis, if one motor moves in one direction while the other moves in the contrary one. Figure 4 shows the algorithm state machine (ASM) for the obstacle avoidance behavior.

There are several options for the physical implementation of state machine's algorithms. One option is to use discrete components, such as flip-flops. Another option is to use standard memories or VHDL programming [4], etc. Figure 5 shows the architecture "Addressing by Trajectory" that implements a state machine with a memory, in it the ASM is encoded in a look-up table that contains the next state and

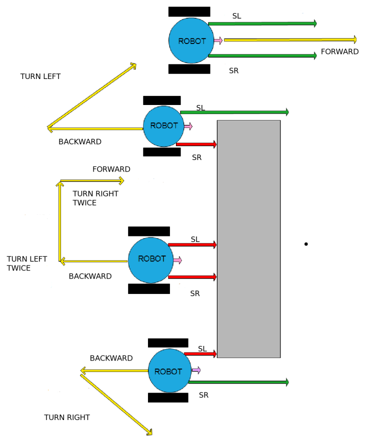

Figure 3. Robot avoiding an obstacle

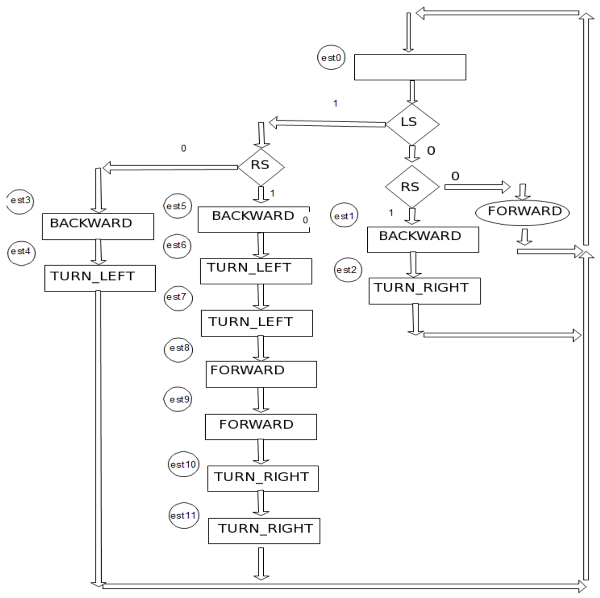

Figure 4. Algorithm State Machine for a mobile robot that avoids obstacles

the outputs of each state. The inputs of the state machine, sensors, and the next state are linked together to form the address of the memory that contains the next state and outputs for the present state. The speed of the state machine is controlled by the clock connected to the register that stores the memory address of the next state and to the register that stores the outputs.

Using this method, depending of the number of states $N_s$, each state is represented by $N_b = \lceil log_2(N_s) \rceil$ bits. The number of memory locations used is $2^{(N_s+N_i)}$, where $N_i$ is the number of inputs. Figure 4 shows the ASM with 12 states and with two inputs. Four memory locations are used to represent each state and the total number of memory

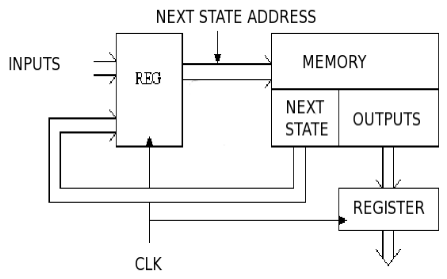

Figure 5. Implementation of an State Machine with the Addressing by Trajectory method

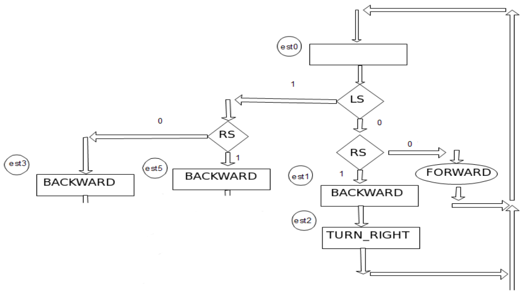

Figure 6. State zero and its next states depending on inputs LS and RS

locations is 64. Table I shows the encoding of state 0, 1, 2, 3 and 5, corresponding to the ASM shown in figure 6. The inputs LS and RS represent the digital value of the infrared sensors, these variables are 1 when there is an obstacle in front of them and 0 when there is no obstacle.

For the outputs, 2 bits are used to indicate the direction $D$ that the robot should follow: 00 forward; 01 turn left; 10 turn right and 11 backward. Another 4 bits are used to indicate the magnitude , $M$, of the command. For translations, $M$ indicates the distance that the robot will move forward or backward. In the small mobile robot used in the experiments the maximum advance was 10 cm, represented with 1111 (correspondingly, 0000 represents no movement). For rotations, $M$ indicates the angle in radians that the robot rotates left or right, $\pi/2$ is the maximum value with 1111.

In state 0 there is one conditional output, FORWARD, shown inside the oval, that is generated when both inputs LS and RS are equal to 0, and it will have a maximum magnitude of 1111, that is, the mobile robot advances 10 cm. For the other cases of LS and RS the robot would stop, thus the magnitude is 0000. For states 1,2,3 and 5, even there are no input variables to sense, still four memory locations are used for each of them. Notice that the magnitude for TURN RIGHT is 0011, meaning that the robot turns to the right $\pi/4$ radians.

| MEMORY ADDRESS | | | MEMORY CONTENT | | |
|---|---|---|---|---|---|
| Present State | Inputs | | Next State | Outputs | |
| ABCD | LS | RS | ABCD | DIR. | MAGNITUDE |
| 0 0 0 0 | 0 | 0 | 0 0 0 0 | 0 0 | 1 1 1 1 |
| 0 0 0 0 | 0 | 1 | 0 0 0 1 | 0 0 | 0 0 0 0 |
| 0 0 0 0 | 1 | 0 | 0 0 1 1 | 0 0 | 0 0 0 0 |
| 0 0 0 0 | 1 | 1 | 0 1 0 1 | 0 0 | 0 0 0 0 |
| 0 0 0 1 | 0 | 0 | 0 0 1 0 | 1 1 | 1 1 1 1 |
| 0 0 0 1 | 0 | 1 | 0 0 1 0 | 1 1 | 1 1 1 1 |
| 0 0 0 1 | 1 | 0 | 0 0 1 0 | 1 1 | 1 1 1 1 |
| 0 0 0 1 | 1 | 1 | 0 0 1 0 | 1 1 | 1 1 1 1 |
| 0 0 1 0 | 0 | 0 | 0 0 0 0 | 1 0 | 0 0 1 1 |
| 0 0 1 0 | 0 | 1 | 0 0 0 0 | 1 0 | 0 0 1 1 |
| 0 0 1 0 | 1 | 0 | 0 0 0 0 | 1 0 | 0 0 1 1 |
| 0 0 1 0 | 1 | 1 | 0 0 0 0 | 1 0 | 0 0 1 1 |
| 0 0 1 1 | 0 | 0 | 0 1 0 0 | 1 1 | 1 1 1 1 |
| 0 0 1 1 | 0 | 1 | 0 1 0 0 | 1 1 | 1 1 1 1 |
| 0 0 1 1 | 1 | 0 | 0 1 0 0 | 1 1 | 1 1 1 1 |
| 0 0 1 1 | 1 | 1 | 0 1 0 0 | 1 1 | 1 1 1 1 |
| 0 1 0 1 | 0 | 0 | 0 1 1 0 | 1 1 | 1 1 1 1 |
| 0 1 0 1 | 0 | 1 | 0 1 1 0 | 1 1 | 1 1 1 1 |
| 0 1 0 1 | 1 | 0 | 0 1 1 0 | 1 1 | 1 1 1 1 |
| 0 1 0 1 | 1 | 1 | 0 1 1 0 | 1 1 | 1 1 1 1 |

Table I
MEMORY REPRESENTATION FOR STATES 0,1,2,3 AND 5, SHOWN IN FIGURE 6

## II. BACKGROUND

### A. 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 [7].

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

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

that forces the robot to move to the direction that the potential field decrees, where $\vec{\nabla}$ 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 filed $U_{rep}$

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

thus

$$\vec{F}(q) = \vec{F}_{atr}(q) + \vec{F}_{rep}(q)$$

where

$$\vec{F}_{atr}(q) = -\vec{\nabla} U_{atr}(q)$$

$$\vec{F}_{rep}(q) = -\vec{\nabla} U_{rep}(q)$$

*1) Repulsive potential field:* The goal of the repulsive potential field is to create a force that take 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,

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

*2) 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

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

in this way the movement in discrete times is defined by

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

where $\delta_i$ 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.

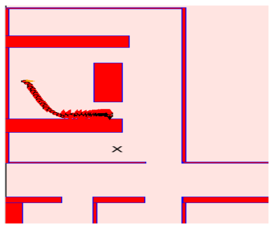

Figure 7. Robot got stuck in a local minimum

Figure 7 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 8 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.

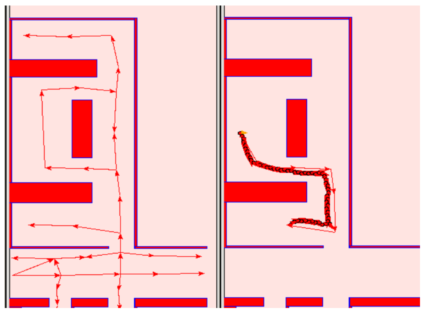

Figure 8. The figure's left side shows the topological map of an environment, the right side shows the path generated by a potential fields approach combined with the Dijkstra algorithm

### B. Genetic algorithms

One of the objectives of our research was how to generate the algorithms of state machines for mobile robots automatically, Wadhe [5] showed that these type of algorithms can be found by GA techniques.

GA are part of evolutionary computing, they are based on Darwin's theory evolution. Systems that use this learning method improve their performance through a process that simulate 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 [3].

*1) 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 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 [6].

*2) 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 offsprings are generated.

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

*4) 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.

## III. *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 a set of next states and outputs, associated to each state, that form the behavior, $B = \{s_1o_1, ..., s_io_i, ..., s_ko_k\}$, that is stored in the memory of the state machine shown in figure 5:

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 represent the next states and the outputs together, $\{s_i, o_i\}$. Depending on the number of states selected, $N_s$, each next state $s_i$ is represented by $N_b = \lceil log_2(N_s) \rceil$ bits. For the outputs 2 bits are used to indicate the direction, $D$, another 4 bits are used to indicate its magnitude , $M$, as it was explained before.

2. Each individual (chromosome) is evaluated giving a fitness value according to the individual's performance. The 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.

Constants $K_1, K_2, K_3, K_4$ and $K_5$ are used to tone 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, 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 $\epsilon$.

## IV. *Simulator*

For each generation the GA needs to evaluate population's individuals, doing this with the real robot it would required to 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 [7]. The sensor value is then the distance between the sensor position and the position of the intersection with a line of the obstacle.

The mobile robot has an array of infrared sensors, one simulation can be seen in figure 9, where the rectangle represents an obstacle, the circle represents the robot and the lines represent the sensors readings.

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

## V. *FPGA Implementation*

The best behavior, found by the GA, that the mobile robot will use to navigate, that is, the algorithm state machine encoded in a look-up table, it is programed in a FPGA using the state machine architecture that implements the addressing by trajectory method shown in figure 5. This state machine controls the Robot's motors and reads data coming from the infrared sensors. The mobile robot that was used to test the system is equipped with two wheels that allows the robot a differential movement.

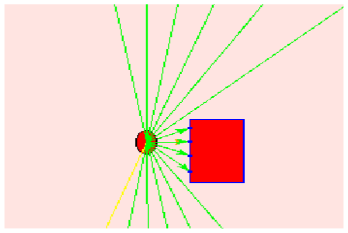
Figure 9. Simulation of the robot's proximity sensors

The frame of the robot consists of an Altera's TerAsic board, the MAX II Micro Kit [8], with four infrared sensors is shown in figure 10.

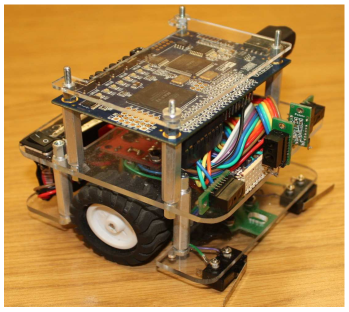
Figure 10. Robot design with its behavior, found by a GA, programed in a FPGA

One of the most complicated part of this project was the matching of the simulated robot with the real one, there were several problems that needed to be address before a successful robot was obtained. One of the problems was the setting of the PWM clocks that control the speed of the robot's motors, that matched the distances that the simulated robot needed to accomplish during training.

## VI. *Experiments and Results*

The system was tested with the environment shown in figure 9.

In figure 9 the red rectangle represents an obstacle of 1.0 x 0.5 m. and the diameter of the robot is 0.10 m. The simulated sensor returns 682 readings that are grouped together to form the inputs of the state machine:

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

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 0.8 of probability of crossover, 0.01 of mutation rate, with 20 generations and using the Vasconcelos GA. The ASM had 8 states; 4 inputs, 2 infrared sensors in front and two in the left and right sides of the robot; 6 outputs, the total memory used to represent the ASM was 128 locations by 9 bits.

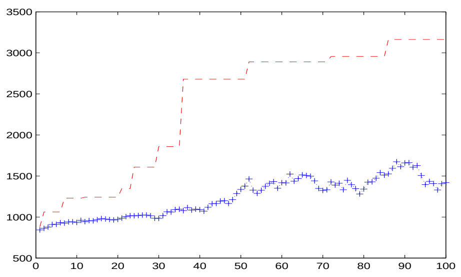

Figure 11. Fitness function of the best individual, -, and population average, +, with 4 infrared inputs

After it was found a set of individuals that go in straight line, they were set in different positions in front of the obstacle, like the one shown in figure 12, to learn how to avoid the obstacle in these positions.

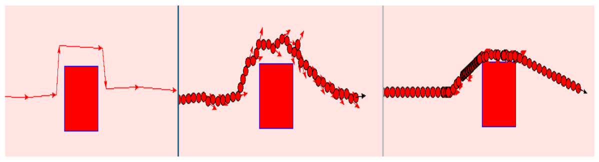

Figure 12. The figure's left side shows the topological map of the environment, the middle side shows the path generated by a potential fields approach combined with the Dijkstra algorithm and the right side shows the state machine's behavior found by the GA

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

In figure 13 shows an individual that solves the problem, it is not the most efficient one, because it chose a long way compared when it was trained in the same position, by changing the constants in the fitness function a better individual could be obtained that would go using the shortest paths.

Then the FPGA was programed with the best individual and its performance fulfill our expectations.

## VII. Conclusions

In this paper we presented how to to implement mobile robots' behaviors using FPGAs, an obstacle avoidance behavior was found, automatically using genetic algorithms, that mimics a potential field navigation approach without having the problem of getting stuck in a local minimum and without using a topological map. The behavior was implemented in a state machine architecture, controlling the overall operation of a small mobile robot, and it was successfully implemented in a programmable logic device, FPGA. We proved that FPGAs are a good option for the implementation of mobile robots' behaviors and that GA is a method for finding them.

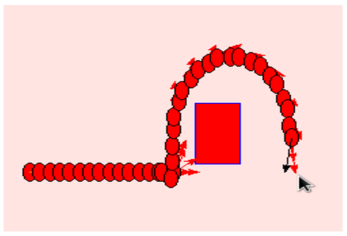

Figure 13. Obstacle avoidance behavior of a robot when it encounters an obstacle in the lower edges of it, during testing.

## Acknowledgment

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

## References

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

[2] Jesus Savage, Marco Morales, (2010). 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] Dario Floreano and Claudio Mattiussi (2008). Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies by Dario Floreano and Claudio Mattiussi, MIT Press, MA

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

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

[6] 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, 25/05/2010

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

[8] Terasic Technologies, (2009). Max II Micro UserManual release v1.32.