Configurable Mobile Robot Behaviors Implemented on FPGA Based Architectures

Jesus Savage, Jesus Cruz, Mauricio Matamoros +, David A. Rosenblueth, Stalin Muñoz, Marco Negrete
BioRobotics Laboratory, School of Engineering
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS)
Universidad Nacional Autónoma de México (UNAM)
+ BioRobotics Laboratory, School of Mechanical
Maritime and Materials Engineering, Delft University of Technology, Netherlands.

[1] ***Abstract*—For educational purposes there is a need to teach electrical and computer engineering students the basics of the design of state machines using programmable logic devices, and for students interested in mobile robots to teach them the basics of mobile robots' behaviors. At the same time one of the topics of interest in the mobile robot's community is how to generate their new behaviors, using state machines, automatically.**

This paper discusses how to create mobile robots' behaviors using genetic algorithms (GAs), implementing them in programmable logic devices (FPGAs) and how these behaviors can be programmed by students in a mobile robotics course.

The behaviors are encoded as algorithm state machines and using feed-forward artificial neural networks (ANN). In the state-machine approach, each individual's chromosome represents, given a set of inputs coming from the sensors and the current state, both the next state and the outputs that control the robot's movements. In the ANN approach, whose weights are found also with GAs, a pipeline architecture was built to perform it; each pipeline executes a layer of the ANN, thus once the pipeline is full the execution speed of the ANN is one sensors' clock cycle. We evaluate the behaviors generated by the GA according to a fitness function that grades their performance for avoiding obstacles. The inputs to such a robot are infrared sensors to detect obstacles; the outputs are the velocities of its wheels. Our objectives are first, to prove that GAs are a good option as a method for finding behaviors for mobile robots' navigation, and second, that these behaviors can be implemented in an efficient way in FPGAs. We tested both behaviors in a small mobile robot, that is build in an electrical engineering course that teaches how to build mobile robots.

***Keywords*-**
Mobile Robots Behaviors; Genetic Algorithms; Artificial Neural Networks; FPGAs

I. Introduction

In our school, in recent years, we have been developing laboratory assignments for small mobile robots which exhibit obstacle avoidance behaviors, based on state machines using FPGAs [1]. These assignments have been tested with mechatronic, electrical and computer engineering students and they considered that the material presented in them was sufficient to understand the basics of how to build state machines using programmable logic devices. The material is also useful for students interested in mobile robots, specifically in the area of subsumption theory using behaviors to control a mobile robot [2].

Since the beginning of the research in mobile robots one of the topics of interest was how to automatically generate their navigation behaviors, using state machines. In some cases this has been done using genetic algorithms and some of them have been implemented in FPGAs.

Also for mobile robots participating in the RoboCup soccer small size league [3], the basic capabilities they should have, are: fast and reliable obstacle avoidance behaviors. These behaviors implemented only on traditional state machines using discrete components as gates and flip-flops lack the precision obtained implementing them using artificial neural networks.

In this paper, we show how to configure obstacle avoidance behaviors for mobile robots using standard algorithm state machines and using artificial neural networks (ANN); both behaviors are generated automatically by GAs. The paper is organized as follows. First there is a description on how to implement robots' behaviors using state machines, then it is described how to use genetic algorithms to generate them, it is explained then the simulator and the FPGA implementation, the test and results are presented and finally the conclusions.

II. Implementation of Behaviors Using State Machines

Figure 1 shows a simple behavior that the robot executes when it encounters an obstacle in different situations. The robot has two sensors on its left and right sides, respectively, allowing obstacle detection. For locomotion the robot has two motors, whose speeds can be controlled.

Figure 2 shows the algorithm state machine (ASM) for the obstacle-avoidance behavior shown in Figure 1. For the output TURN_LEFT the robot turns 45 degrees, for TURN_RIGHT the robots turns -45 degrees. Thus, for turning 90 degrees, two states turning 45 degrees in each of them are used.

[1]Acknowledgments: This work was supported by PAPIIT-DGAPA UNAM under Grant IG100915, PASPA-DGAPA and Conacyt grants 221341 and 261225.

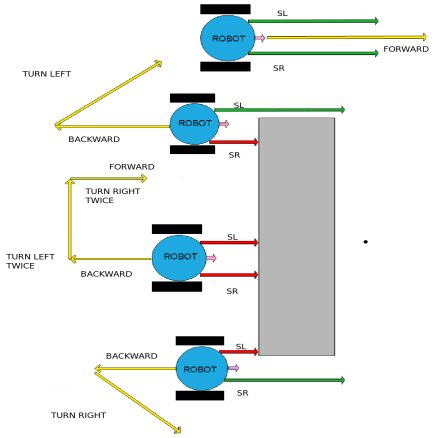

Figure 1. Robot avoiding an obstacle

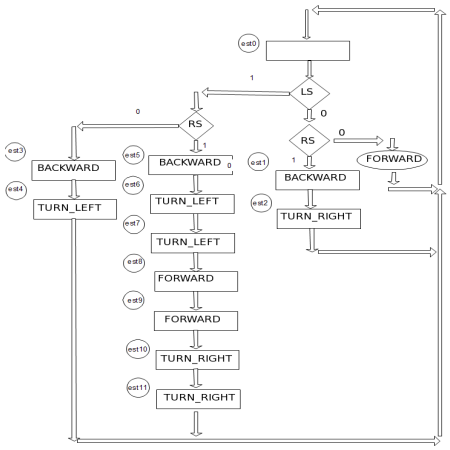

Figure 2. Algorithm State Machine for a mobile robot that avoids obstacles

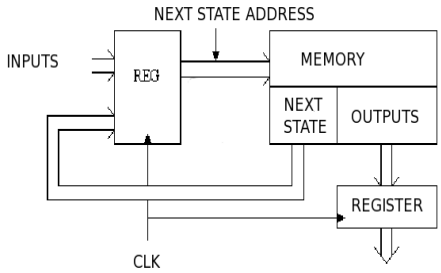

Figure 3. Implementation of an State Machine Using Memories

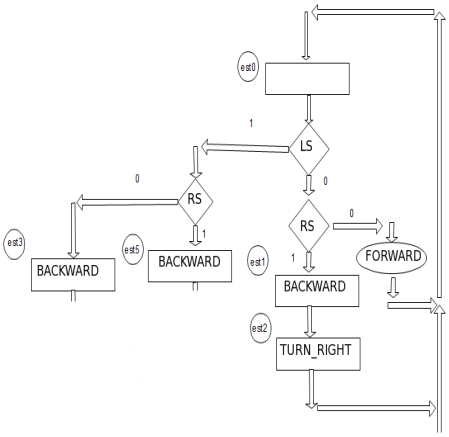

Figure 4. State zero (est0) and its next states depending on inputs LS and RS

There are several options for the physical implementation of state machine's algorithms [4] in small mobile robots similar to the ones used in the category small size in the RoboCup competition. Since speed is an issue, several small-size teams have recently opted for a combination of microcontrollers together with FPGA devices [5]. Thus, there are simple architectures that implement algorithm state machines using look-up tables. Figure 3 shows an architecture of a state machine that uses a memory. In such an architecture, the ASM is encoded as a look-up table containing the next state and the outputs of each state. The inputs of the state machine, i.e., the 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 on 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 2 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 locations is 64. Table I shows the encoding of state 0, corresponding to the ASM shown in Figure 4. 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, two bits are used to indicate the movement direction MD that the robot should follow: 00 forward; 01 turn left; 10 turn right and 11 backward. Four more bits are used to indicate the magnitude MAG of the command. For translations, MAG 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, MAG indicates the angle in radians that the robot rotates left or right, $\pi/2$ is the maximum value with 1111.

Table I shows part of the look-up table for algorithm in Figure 2. Shown is the coding of state 0, where there is one

MEMORY ADDRESS			MEMORY CONTENT		
Present State	Inputs		Next State	Outputs	
ABCD	LS	RS	ABCD	MD	MAG
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
...					

Table I
MEMORY REPRESENTATION FOR STATE 0, SHOWN IN FIGURE 4

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.

One of the goals of this research is to create mobile-robot behaviors, that is, navigation algorithms, automatically. By changing some of the bits of the look-up table of the state machine architecture shown in Figure 3 the behavior of the robot can be modified, thus we proposed the use of GAs to obtain an optimum behavior by generating look-up tables with new behaviors.

A. Implementation of Behaviors Using Artificial Neural Networks

Like the biological archetype, an artificial neuron has an arbitrary number of inputs and one output. All inputs x_i are multiplied by their weight value w_i and summed up, supplemented by the bias value θ. The sum is then propagated via a transfer function f to the output y.

There are several different possible transfer functions, but in this work an approximation to the Sigmoid Function will be used. Functions in the sigmoid class are continuous, differentiable, monotone and have a limited co-domain, usually in the range of $[0; 1]$ or $[-1; 1]$.

B. Network Architectures

There exists a wide range of neural network types. Their differences are based on the structure of the network as well as on the learning method used.

Here, we use the Feed-Forward network, which is perhaps the most used one, and the one with most applications. This type of network can also be found under the name of Multi-Layer Perceptron Network.

The performance of neural networks originates from the connection of individual neurons to a network structure that can solve more complex problems than the single element can. Feed-forward networks organize the neurons in layers. The output of one neuron is then:

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

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.

Connections are only allowed between neurons in different layers and must be directed toward the network output. Connections between neurons in the same layer are prohibited. The output layer is the last layer in the network and provides the network outputs.

A robot's behavior to avoid obstacles can be implemented using an ANN. Such ANN has as inputs the sensors that detect obstacles and its outputs control the speed of the robot's motors. The weights of the ANN are found in a way that the ANN learns how to avoid obstacles. In this research the weights of the ANN are found using also GAs.

III. GENETIC ALGORITHMS

Wahde [6] showed that it is possible to find behaviors for mobile robots by GA techniques.

GAs are part of evolutionary computing, and are based on Darwin's theory of evolution. Systems that use this learning method improve their performance through a process that simulates reproduction by the survival of the fittest individuals. First, a population of individuals is created, represented by a set of chromosomes, which consists 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.

A. Genetic Algorithm for Finding Navigation Behaviors Using State Machines and Neural Networks

A GA is used to automatically generate a navigation behavior, that would allow a mobile robot to navigate in an environment avoiding obstacles.

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

- For the behaviors using a state machine architecture: The GA finds a set of next states and outputs, associated with each state, that form the behavior: $B = \{s_1 o_1, \ldots, s_i o_i, \ldots, s_k o_k\}$:
  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 two bits are used to indicate the direction, MD, another 4 bits are used to indicate its magnitude , MAG , as it was explained before.

- For the behaviors using an ANN, the chromosome represents the weights of it: $B = \{w_{11}^1, \ldots, w_{1I}^1, \ldots, w_{11}^m, \ldots, w_{1J}^m, \ldots, w_{11}^n, \ldots, w_{1K}^n\}$ The string is separated into small segments that represents the neural network weights, $\{w_{ij}^k\}$, that is, coded using a binary number $\{(0011010\ldots011)\}$ 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 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 for individual i is:

$$F(i) = 1/(DistDest_i+1.0)+1/nSteps_i+1/(nStuck_i+1.0) + 1/(nBack_i + 1.0)$$

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; $nBack_i$ is the number of times the robot went backward.

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

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

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

IV. *Simulator*

For every generation the GA needs to evaluate each individual, doing this with a real robot it would require so much time, that would be intolerable to do this.

Thus, the GA needs a simulator, as close as it can be to simulate well the real robot and its environment.

The simulator reads the individuals chromosomes and executes the algorithm state machine represented by them, simulating the movements of the robot, as well as the values of its 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.

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 5.

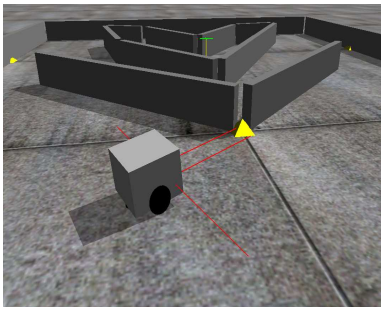

Figure 5. Simulation of the robot's proximity sensors

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

V. *FPGA Implementation*

A. *State Machines using FPGAs*

The best behavior, the algorithm state machine encoded in a look-up table, is programmed in a FPGA using the state machine architecture shown in Figure 3. The mobile robot that was used is equipped with two wheels that allows the robot a differential movement. The frame of the robot consists of an Altera board, with four infrared sensors is shown in Figure 6.

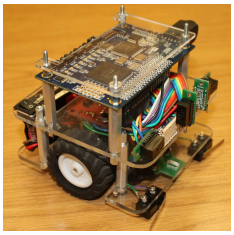

Figure 6. Robot design with its behavior, found by a GA, programmed in a FPGA

One of the most complicated parts of this project was the matching of the simulated robot with the real one. There were several problems that needed to be addressed 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 simulation.

B. *ANN using FPGAs*

We designed and implemented a versatile architecture for the implementation of a multipurpose simple feed-forward artificial neural network (ANN) with n neurons and m hidden layers. This architecture consists of a clone of a microcontroller, 6811, with a modified set of instructions

to operate a special proposed ANN module. Figure 7 shows the block diagram of the architecture.

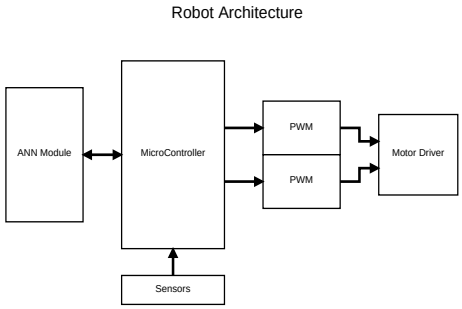

Figure 7. Robot architecture with an ANN module.

The ANN module is built using a pipeline architecture, see Figure 8, in which each stage of the pipeline executes a layer of the ANN, thus once the pipeline is full the execution speed of the ANN is one sensors' clock cycle.

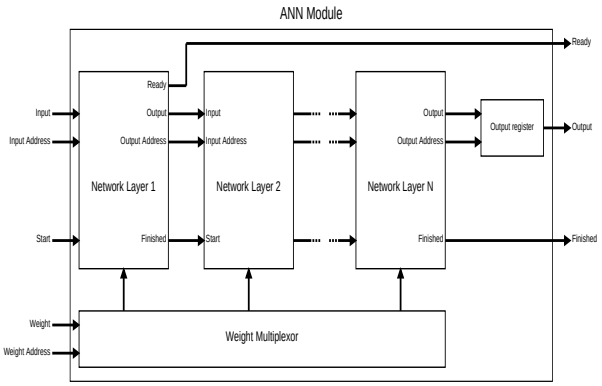

Figure 8. Feed Forward ANN.

Each pipeline stage contains an ANN layer, shown in Figure 9, that contains the weights, inputs and Neuron module.

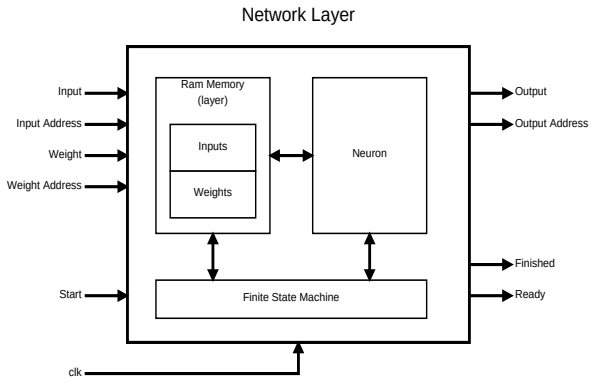

Figure 9. ANN network layer

This architecture can perform 32 bits floating point operations for the calculation of the ANN outputs, and can be easily extended and configured to operate with any number of neurons and only limited by the hardware requirements. Due to the FPGA hardware limitationsthe design has only one Neuron module per layer, where a finite state machine (FSM) control the neuron's inputs and outputs in every state for the calculation of the layers' outputs, making this approach efficient in terms of hardware cost.

Neuron Module

This module is composed of a 32-bit floating point adder, a 32-bit floating point multiplier, a 32-bit floating point divider and a finite state machine that selects the inputs and outputs to calculate an output, as seen in Figure 10.

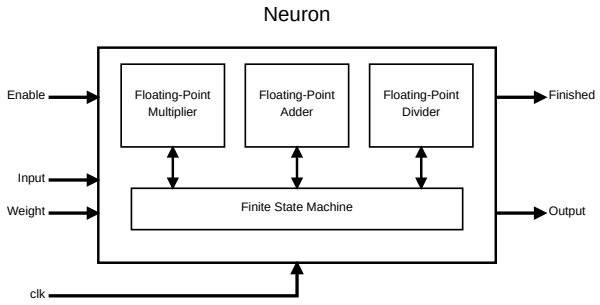

Figure 10. Components of the Neuron module.

The Neuron module calculates the accumulation of the inputs and weights multiplications. When all the inputs–weights pairs have been added, it calculates the output using an activation function. As activation function it was used the *fast sigmoid approximation* where $f(x) = \frac{x}{1+|x|}$, approximates the commonly used hyperbolic function $f(x) = tanh(x)$. This function needs only one floating point division and the absolute value of the input, making this function a desirable by the hardware limitations. This way, a single neuron can calculate all the outputs of the layer, regardless of what is occurring in other stages, allowing a pipeline execution. For the implementation of the whole network the outputs of the previous layer are connected to the inputs of the next layer, as seen in Figure 9. The inputs of the first layer are the sensors' values, the outputs of the final layer are read by the microcontroller and it sends them to the motors. Two Pololu infrared digital sensors were used to detect obstacles. The output of these sensors is digital, and it has a high value if the sensor could not detect something, if detects something the output is zero, and it detects objects in the rage of 2 to 10 cm. The sensors were connected directly to the FPGA board without the need of any pre processing to the signal. The robot has two DC Pololu, with a quadrature encoder, their speeds were controlled using PWM using a dual full bridge driver.

VI. *Experiments and Results*

A. Implementation of Behaviors Using State Machines

Figure 11 shows one of the simulated environments used to evaluate the individuals generated by the GA. This environment has a delimiting wall and several objects, represented as red polygons, that are placed randomly in it, the number of sides of each polygon also is a random number. Between each polygon is a margin allowing the robot to pass through.

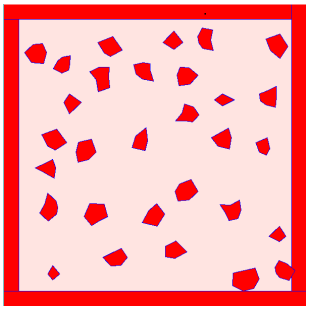

Figure 11. Testing Environment for the state machine approach.

The size of the population was 100 individuals, with 0.8 of probability of crossover, 0.01 of mutation rate, with 1,000 generations. The ASM had eight states; four inputs, two infrared sensors in front and two in the left and right sides of the robot; six outputs, the total memory used to represent the ASM was 128 locations by 9 bits. Figure 12 shows the fitness function of the best and worst individual and the average of the population for each generation. Then the FPGA was programmed with the best individual and its performance was according to the simulation. Because of the hardware implementation simplicity the clock time to calculate a new value is 200 ns.

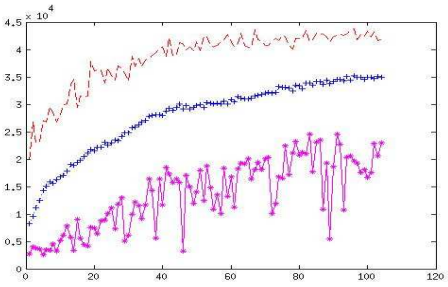

Figure 12. Best individual fitness shown in red color, average population in blue and worst individual in pink.

B. Implementation of Behaviors Using Artificial Neural Networks

For the feed-forward artificial neural networks behavior the GA used a population of 100 individuals, with 0.9 of probability of crossover, 0.10 of mutation rate, with 200 generations. The target neural network had 10 distance sensor inputs, 4 hidden neurons and 2 outputs, which correspond to the robot's linear and angular velocities. Once the ANN pipeline is full the clock time to calculate a new value of the ANN is $8.6\mu s$. Videos showing the performance of the robot with the state machine, ANN approaches and simulation can be seen in the following link: http://biorobotics.fi-p.unam.mx/media/youtube

VII. Conclusions

In this paper, we presented how to generate mobile robots' behaviors using GA. Obstacle avoidance behaviors were found automatically using genetic algorithms and they were successfully implemented in a programmable logic device, FPGA. Two different obstacle avoidance behaviors were implemented: one uses state machine architecture and the other uses a feed-forward ANN, controlling the overall operation of a small mobile robot. The state machine approach is 430 times faster than the ANN one, but the FPGA space used doubles each time a new input is added. Also the robot movements are less accurate than the ones generated by the ANN. We proved that GA is a good option as a method for finding behaviors for mobile robots' navigation and also that these behaviors can be implemented in a FPGA. From the educational point of view, these techniques were successfully implemented for engineering students in a mobile robotics course in our university.

References

[1] Jesus Savage, Marco Morales, *Laboratory Assignments to Teach the Basics of Programmable Logic Applied to Mobile Robots*. Primer Taller de Cómputo Reconfigurable y sus Aplicaciones en Educación e Ingeniería, ReConFig 2010, Cancún, Quintana Roo, México.

[2] Brooks, R. A. *A robust layered control system for a mobile robot*. IEEE Journal of Robotics and Automation, RA.-2(1):14-23a, 1996.

[3] RoboCup Small Size Category, http://robocupssl.cpe.ku.ac.th/ 2014.

[4] Juan Liu, Zhiwei Liang, ZJUNIlict: RoboCup 2013, Small Size League Champion. RoboCup 2013: Robot World Cup XVII (Lecture Notes in Computer Science), Springer, 2014.

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

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

[7] Dario Floreano and Claudio Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies* by Dario Floreano and Claudio Mattiussi, MIT Press, MA, 2009