Robotics State Machine Behaviors Derived with Genetic Algorithms

Jesus Savage, Stalin Muñoz, Marco Negrete and Carlos Rivera
Bio-Robotics Laboratory, School of Engineering,
Universidad Nacional Autónoma de México (UNAM),
robotssavage@gmail.com

***Abstract*— In this work, we propose the use of genetic algorithms to evolve finite state machines (EFSM) to generate reactive behaviors for mobile robots . We build these models using a modified version of FSMs and we propose an optimization process using genetic algorithms to find the best behavior for a mobile robot to avoid obstacles while it goes to a destination. We first train our EFSM in a simulated environment and then test the best model in a number of unknown scenes. We compare our algorithm with a human made FSM, where we show that the evolutionary approach outperforms it.**

I. Introduction

Behaviors are commonly used for mobile robots navigation [1] and they can be implemented using algorithm state machines (ASM). Figure 1 shows an example of a behavior for avoiding obstacles: a robot with two sensors S_L and S_R (left and right) to detect obstacles that performs four different movements depending on what is detected. Figure 2 shows the corresponding ASM to implement this behavior. There

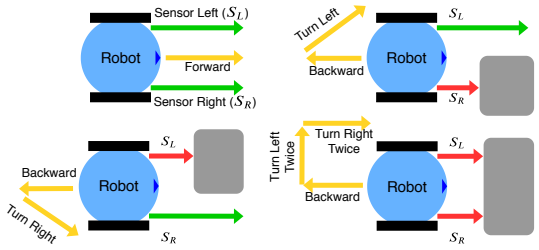

Fig. 1. Obstacle avoidance behavior.

are several options for the physical implementation of ASMs. In this work we propose an architecture (shown in figure 3) based on standard memories. In this architecture, an ASM is encoded in a look-up table that contains the next state and outputs for each state. The ASM inputs (given by sensors) and the next state are concatenated to form the memory address that contains the next state and outputs for the present state. Using this method, each state is represented by $N_b = \lceil log_2(N_s) \rceil$ bits where N_s is the number of states. The number of memory locations needed is given by $2^{(N_s+N_i)}$, where N_i is the number of inputs.

Figure 2 shows an ASM with 12 states and 2 inputs. For this ASM, four memory locations are used to represent each state and the total number of memory locations is 64 (upper power of two). Table I shows the memory data needed to

*This work was supported by PAPIIT-DGAPA UNAM under Grant IG-100818

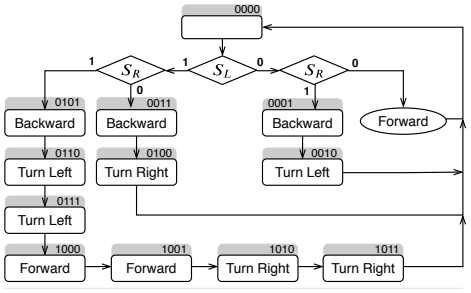

Fig. 2. ASM for a mobile robot that avoids obstacles.

encode the state 0000 of such ASM. Inputs S_L and S_R represent the digital values of sensors, whose state is 1 when there is an obstacle in front of them and 0, otherwise. ASM outputs are encoded with 6 bits. 2 bits are used to indicate the direction (DIR) 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 of the command: for translations, it indicates the distance that the robot will move forward or backward and for rotations, indicates the angle that the robot rotates, with a maximum value of $\pi/2$ [rad] corresponding to 1111. In state 0, the conditional output *Forward* (shown inside the oval in figure 2) is generated when both inputs S_L and S_R are equal to 0. For the other cases the robot will stop, thus the magnitude is 0000.

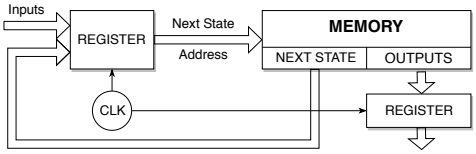

Fig. 3. Implementation of an ASM using memories.

As we can see in table I, if some of the bits in the memory content of this table are changed, then the robot behavior will also change. The main goal of this work is to find the best configuration of the memory content to get a behavior that avoids obstacles while it tries to reach a destination which, in this case, is given by a light source.

II. Genetic Algorithm for Finding Navigation Behaviors

In this work we use a genetic algorithm (GA) [2] to automatically generate a navigation behavior (given by an ASM)

MEMORY ADDRESS			MEMORY CONTENT		
Present State	Inputs		Next State	Outputs	
ABCD	S_L	S_R	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

TABLE I

MEMORY DATA FOR STATE 0 SHOWN IN FIGURE 2.

that allows a mobile robot to navigate in an environment avoiding obstacles. The GA finds a set of next states and outputs, associated to each state, that form the behavior: $B = \{s_1o_1, ..., s_io_i, ..., s_ko_k\}$. The robot's behavior is encoded in the look-up table content as the one shown on table I. General steps for evolving behaviors are:

1. 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 N_s, each next state s_i is represented by $N_b = \lceil log_2(N_s) \rceil$ bits.

2. Each individual (chromosome) is evaluated with the following criteria: distance between goal and last position reached (d_g), number of times the robot hits an obstacle (n_o), number of steps needed to reach the goal (n_g) and the number of times the robot went backwards (n_b). The proposed fitness function for the i-th individual is given by

$$F_i = \frac{K_d}{d_g + 1} + \frac{K_o}{n_o + 1} + \frac{K_g}{n_g + 1} + \frac{K_b}{n_b + 1}$$

where constants K are chosen according to the desired robot performance. For instance, if it is desired that the final robot's behavior goes backward only few times during navigation, then K_b should be several times bigger compared to the other constants.

3. Best individuals are selected according to their fitness function and a new population is generated through evolutionary operators (selection, crossover and mutation).

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

5. Steps 2 to 4 are repeated for N generations or until the difference of the overall fitness criteria between two generations is less than a given ϵ. Resulting behavior is an Evolved Finite State Machine (EFSM).

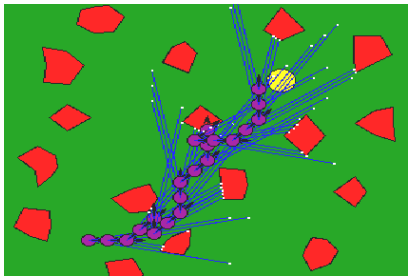

Fig. 4. EFSM tested on a simulated environment.

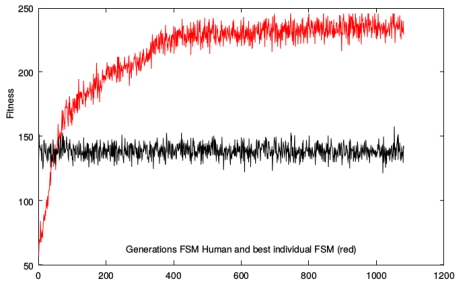

Fig. 5. Fitness of the EFSM (red) and the Human FSM (black).

III. EXPERIMENTS AND RESULTS

The algorithm to avoid obstacles was tested in a simulated environment shown in figure 4, where the goal destination is described by the yellow circle, free space is colored green, obstacles are described by polygons in red, robot is the purple circle and the proximity sensor readings are represented by blue lines.

Sensor readings are simulated by calculating the distance between the robot and objects in front of it. Gaussian noise is added to these values. Each EFSM has 16 states (4 bits); two obstacle avoidance input sensors (2 bits); one sensor that indicates the intensity of a light source (1 bit) and one sensor that indicates the quadrant where a light source is located (two bits). The EFSM has 6 outputs (3 bits). The total memory used to represent the algorithm behavior was 512 locations each one with 7 bits. Thus, it is required 3584 bits per individual's chromosome.

For the genetic algorithm, we had a population of 100 individuals and the evolution lasted 1,000 generations. Each generation was tested with 10 different environments with an area of $4m^2$ and 30 random obstacles. For each environment, behavior was tested with 20 origin/destination points. In total, 10,000 environments were created with 200,000 origin/destination paths that the simulated robot needed to follow.

To compare the performance of the EFSM, we used as baseline a FSM tuned by a human. Figure 5 shows the fitness functions for each generation of the human FSM and the best EFSM evolved by the GA. As we can see the best performance is done by the EFSM generated by the GA.

IV. CONCLUSIONS

An obstacle avoidance robot's behavior was found automatically using genetic algorithms, we proved that GA is a good option as a method for finding behaviors for mobile robots' navigation. Our experiments proved that the EFSM outperforms the human FSM.

REFERENCES

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

[2] 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