A Simulator using Classifier Systems with Neural Networks for Autonomous Robot Navigation

Lubnen N. Moussi, Fernando J. Von Zuben, Ricardo R. Gudwin, Marconi K. Madrid
DSCE/DCA – FEEC – UNICAMP
Av. Albert Einstein, 400 – Cidade Universitária Zeferino Vaz, Campinas, SP, Brazil
E-mails: lubnen@dsce.fee.unicamp.br , vonzuben@dca.fee.unicamp.br, gudwin@dca.fee.unicamp.br, madrid@dsce.fee.unicamp.br

Abstract: This paper presents a simulator that was developed to assist in the process of implementing high-level autonomous robot navigation algorithms and in the related experimentations. Classifier systems are designed here, using neural networks as classifiers, to perform autonomous navigation. We propose a powerful simulator using classes and objects to be easily updated and extended. The simulator carries a class composed of methods for differential wheels steering, for detecting collision, and for sensor readings. Another class allows the specification of geometric shaped objects, which can also be detected as obstacles in the environment. In addition, operators are available to deal with credit assignment, genetic algorithms, and inference of the classifiers. By designing and constructing the simulator, we create conditions to explore the potentialities of neural networks as classifiers.

I. INTRODUCTION

Classifier systems (CS) [2] were originally proposed as learning devices associated with an evolutionary algorithm for rule discovery. Each classifier corresponds to a propositional rule, usually encoded as a chromosome of bits. Populations of classifiers operate in parallel, being evaluated individually by a credit assignment mechanism, according to the system performance after taking a sequence of decisions (proposed by the classifiers selected to act on the environment). This original version of CS was proposed by Holland [4] with the purpose of modeling natural evolutionary processes, presenting flexibility of operation and mechanisms of structural adaptation that few intelligent systems have shown till now.

CS using neural networks as classifiers have been demonstrated to perform properly when applied to mobile autonomous robot navigation [6]. Moussi *et al.* [6] used a prototype of a computational simulator, and here a more elaborated simulator will be presented, including additional conclusions derived from the new experiments. The results will indicate that it would be worth to proceed with a deeper investigation about new perspectives for classifier systems.

A brief explanation of classifier systems and the role of neural networks in this context are presented in section II. Section III is dedicated to the Simulator, conceived as a modular structure to implement high level planning for mobile robots. The software platform adopted here was MATLAB, a dedicated programming environment, with all facilities to deal with applied mathematics. Within MATLAB, we explore the availability of classes and objects, and the possibility of generating code using the C++ language. In Section IV, we enumerate distinctive aspects related to the use of neural networks as classifiers, including the increment in the role of each classifier. The universal approximation capability of neural networks [5] improves the flexibility of each classifier, without reducing the learning ability. When compared with traditional rule base classifiers, a significant reduction in the number of classifiers may be observed, without degrading the performance of the system as a whole, because each classifier may become responsible for multiple tasks, not necessarily associated with similar operating points of the navigator.

Section V outlines the concluding remarks and points out relevant considerations for the studies to come.

II. CLASSIFIER SYSTEMS WITH NEURAL NETWORKS

Classifier systems with neural networks are implemented replacing the rule base classifiers, in conventional classifier systems, by neural networks. Some additional references for conventional classifier systems are Goldberg [3] and Richards [8], which give detailed explanations of concepts and parameter initialization mechanisms. In what follows, we will present the most relevant basic concepts.

A. Conventional Classifier Systems (CS)

Classifier systems (CS) are indicated for non-stationary environments with the presence of noise and irrelevant data. They are implemented to operate continuously and in real time, and they pursue implicit goals with sparse rewards. Originally, they refer to a methodology for creating and updating rules (the classifiers), which encode alternative specific actions according to features of the problem in hand.

The Classifier System is composed of a population of classifiers or rules. Associated to each classifier there is a "strength", used to express the energy or power of each classifier during the evolution process (Table 1).

The classifiers are composed of an antecedent and a consequent part. The antecedent part of the classifier is a string of fixed size composed of elements of the ternary alphabet set {0,1,#}. The symbol "#" known as the "don't care" symbol, can assume value "1" or "0", during the comparison with the message from the environment. The consequent part of the classifier is generally given by a string of fixed size composed of elements of the binary alphabet set {0,1}.

The Classifier System communicates with the environment through its message detectors or input interface (Figure 1). These detectors are responsible for the preprocessing of messages, i.e., strings of 0s and 1s. The system acts on the environment through its effectors or output interface (Fig-

ure 1), which decodes the system proposed actions. The appropriate reward applied to the active classifier is determined by the nature of the chained consequences of each action (environment's feedback).

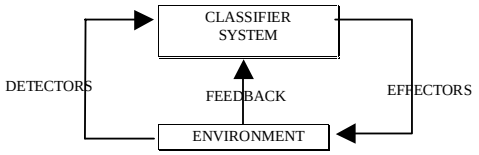

Figure 1 – Classifier System and Environment Interaction

The matching of the classifier's antecedent part with the environment's message defines which classifiers will compete. The competition in this case is based on the strength of the selected classifiers. In the example of Table 1, classifiers **B** and **D** have a higher probability of winning the competition, once selected to compete, due to the level of strength they have.

Another important concept is the "specificity" of each classifier, which is a measure inversely proportional to the quantity of symbols "#" on the classifier's antecedent part.

TABLE 1 – SET OF CLASSIFIERS

Hypothesis	Classifiers or Rules	Strength
A	1#1## : 11	8,5
B	1110# : 01	15,2
C	11111 : 11	5,9
D	##0## : 10	19,0

For example, classifiers **A** and **D** are less specific, being able to match a greater number of messages from the environment. Suppose that each bit from the message 10100, of size S = 5, characterizes an information from the environment. Classifier **A** would match with this message and other 7 messages. However, classifier **C** could match only the message 11111.

The classifier's consequent part is separated from the antecedent part by the symbol " : " and its value determines the action to be applied to the environment through the "effectors". For example, in the case of classifiers **A** and **C**, the sequence 11 could mean "go ahead", determining one among 4 possible movements of an autonomous agent in an unexplored environment.

Classifier Systems are divided into three interactive distinct sub-systems: the Rule and Message Sub-System, the apportionment of Credit Sub-System and the Rule Discovery Sub-System (Figure 2).

A.1 Rule and Message Sub-System

When the message detectors perceive the presence of any message from the environment, this message is sent to the Rule and Message Sub-System (see Figure 2).

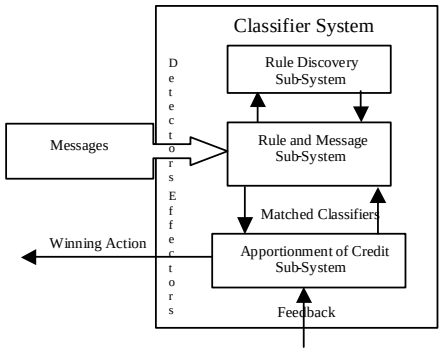

Figure 2 – Simplified Flow (Classifier System/Environment)

The sub-system encodes the message in a way that the Classifier System can recognize. From this moment on, all the classifiers try to match its antecedent part with the message (comparison phase). This matching can be made by a bit-to-bit comparison, according to specific rules, or just by calculating a variation of the Hamming Distance.

Each individual or classifier that matches with the environment message will be sent to the Apportionment of Credit Sub-System.

A.2 Apportionment of Credit Sub-System

During this phase, all classifiers that match with the environment message will participate in a competition. The winner will be allowed to act on the environment.

Several taxes are collected from all individuals of the population: life tax, participation in the competition, and action on the environment.

The environment will reply in response to the action proposed by the winner classifier, providing a feedback to the Classifier System (see Figure 2). It is the responsibility of the Apportionment of Credit Sub-System to incorporate the value calculated, based on the feedback of the environment, to the strength of the active classifier at that moment.

Once the feedback is received from the environment and the credit is attributed to the winner classifier, a new message will be provided by the environment, describing its actual state. Then, once again the message is worked by the Rule and Message Sub-System. The process continues for one epoch of iterations. At the end of each epoch, the Classifier System will take part in another process of evolution, with the discovery of new rules.

A.3 Rule Discovery Sub-System

At the end of each epoch of iterations, the genetic operators are applied to produce the next generation of rules. The objective of the evolutionary process is to search for a Classifier System capable of having an effective interaction with the environment.

Basically, the genetic algorithm chooses the classifiers with greater strength and promotes the reproduction between them, applying the genetic operators of crossover and mutation. The generated children will be introduced into the popu-

lation at the next generation, replacing the weakest individuals [8].

B. Classifier Systems with Neural Networks (CSNN)

In classifier systems with neural networks (CSNN), each classifier is replaced by two neural networks (see figure 3). The antecedent will be represented by a neural network that evaluates the degree of matching with the input message, and the consequent with a neural network that indicates the action.

The output of the Evaluation Neural Network replaces the matching and specificity of the conventional classifier, and the output of the Action Neural Network will perform the same role associated with the consequent of a rule.

The remaining mechanisms are similar to those associated with the conventional CS, and in Section IV we will point out relevant differences between the two approaches.

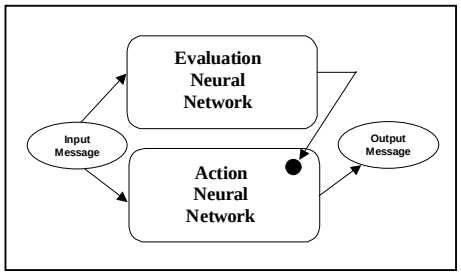

Figure 3 - Neural Networks replacing each classifier.

III. THE SIMULATOR

A. Why simulate?

Though not revealing all the details of a real world application, computer simulation may be effective in indicating the potentialities of several approaches for autonomous navigation, in terms of both theoretical and practical considerations. Simulation may also reduce experimentation time, in the sense that it's possible to run the simulation model faster then the real model. Schultz & Grefenstette [9] and Ramsey *et al.* [7] have shown that knowledge acquired from learning processes under computer simulation is robust and might be applicable to the real world if the simulation is more general, incorporating noise and a diversified set of environmental conditions.

B. Preliminary decisions

First of all, the authors are conscious about the existence of software packages for simulation of autonomous navigation. As an example, we mention "Webots 3.0.1", from Cyberbotics, and directed to the simulation of differential wheels steering robots in a VRML environment [10]. However, it's not compatible (or probably not directly usable) with Borland's or Microsoft's C++ IDE series, what means that we could not have access to many libraries, including MATLAB's. Besides, we were interested in revealing the details of the robots project, i.e. the type of guidance, the type and characteristics of the sensors, and so on.

Back to our simulator, we aimed at employing a familiar and easy-to-use language, that is well accepted and that presents a powerful graphical interface. So we choose MATLAB. It works with classes, which is required for our modular approach, developing a simulator that can be extended to many types of robot guidance, detectors and algorithms for navigation. A restriction one might point out is the fact that MATLAB is an interpreted language, what usually means a bad, or, at least, not so good performance. Fortunately, the amount of computation necessary to simulate the navigation under the control of a classifier system is not so intense, and a conventional computer, with a processor with clock of 300 MHz or higher proved to be enough. Better performance will be required when we start putting more than one robot in the environment. Besides the availability of higher capacity processors, we can consider compiling MATLAB's function in C++.

C. The simulator

The simulator deals with a 2D environment with any topology. We consider differential wheels steering robots as objects of the class *RobotMove*, with a circular shape, according to one of ours goals of simulating the Khepera scientific robot. The robot can have any number of sensors, such that position, aperture and maximum range can be specified. The modular approach allows the use of other forms of guidance and sensors. One method of great relevance for the *RobotMove* class is *RobotDetect*, used to check sensor detection and robot collision.

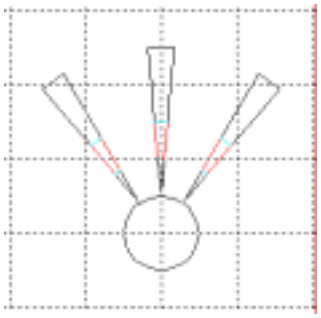

Figure 4 - Illustration of a RobotMove Object

D. Details

Let's take a closer look at the characteristics of the simulator.

1) RobotMove Class: The syntax to create an object *RobotMove* is: **u = RobotMove(Axis,XIn,YIn,TetaIn,DataSensor);** where *Axis* defines the axis, *XIn* and *YIn* stand for x and y coordinates of the center of the robot, *Tetain* is its angular orientation and *DataSensor* is a line vector defining the robot's sensors. Let's consider the following data for the sensors: D**ataSensor = [35 10 100 0 10 100 -35 10 100],** which represents a robot like the one shown in Figure 4. We can notice the

area of perception of three sensors according to the vector **DataSensor**, the first one 35 degrees to the left of the robot heading, the second in the direction of the movement and the third 35 degrees to the right. All of the sensors have an aperture of 10 degrees and maximum range of 100 pixels. The *RobotMove* Class utilizes a private method *Sensor* to define the sensor type, which can detect, through the *Detect* method to be explained in what follows, objects in three ranges: near, middle distance and far.

DWS method handles the mathematics for differential wheels steering, which, in this model, considers the kinematics without acceleration, assumption that fits our purposes by now. Its input parameters are the robot current position and orientation data, the time interval *dt* that will be elapsed and the right and left wheels velocities. The output parameters will give the new robot coordinates and data for plotting.

set method is utilized to settle new properties to the object and to update the position. s*et* has to be used after calling *DWS*. There is one occurrence that might require some care, depending on *dt* and *velocities* of the wheels: when the object overlaps, that is, invades obstacles or the environment contour. In reality, the difference between touching and invading is subtle. It depends on the experiment requirements. Anyway, visually we might observe invasion, and if it is not to occur, we should decrease *dt*.

Data2plot method gives access to the object properties. This method for the RobotMovel is relevant in the case we have more them one RobotMovel object for one of them being able to detect the other. It's done by means of *Detect* that will call *Data2plot* of the other robot to get its data for detection.

Detect is the method that will verify whether the *RobotMove* object touched (that could be invaded) an object, which also could be another robot, or the environment contour. It also gets the sensors readings. It requires as input all the objects plus the environment contour to be detected. *Detect* requires the method *Data2plot* of the object to be detected.

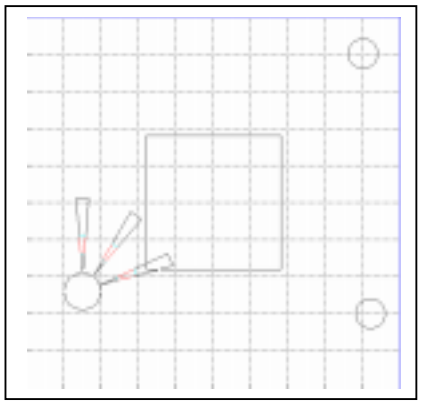

Figure 5 - The environment with a robot and some objects.

2) GeometricForm class: this class permits the creation of geometric shaped objects and their plotting in the robots environment. By now we developed two types of objects: with a circular and with a rectangular shape. The *NewCoordinates* method calculates the new properties for the object in a new position defined by its new center. After defining new properties we have to settle them using the *set* method, which will also show the object in the new position. The *Data2plot* method will be used by the *Detect* method of the *RobotMove*. Figure 5 presents the environment with the robot of Figure 4, a rectangular contour, a central rectangle and two circular objects. The grid is present to indicate the environment measurements.

3) Functions: we implemented four functions: *GeneticAlgorithm*, *Actuator*, *Evaluator* and *PunishmentReward* that handle almost all the learning algorithm for the autonomous navigation. These functions are ready to be implemented as methods of the *RobotMove* class; they will be of relevance as methods when we start to use more then one robot in the environment. *GeneticAlgorithm* applies a basic Genetic Algorithm to classifiers with greater strength to create a new generation that will replace the weakest individuals in the current population. *Actuator* takes the Neural Network of Action of the winner classifier to process the information from the sensors, getting its output as guidance values. A multilayer perceptron with one intermediate layer is considered, with the number of neurons in the first layer being automatically adjusted in accordance with the number of sensors and the number of regions of detection. The number of neurons in the intermediate layer is an input parameter. In the third layer, there is one output for the evaluation network, and, for the action network, as many outputs as are the number of variables required to control the robot.

Evaluator and *PunishmentReward* do the credit assignment and message classification, based on the criteria presented by Richards [8]. There are three main differences outlined in what follows.

The first difference is that the winner doesn't pay the bid tax; only its bid is deducted from its strength. The second is that when there is no change in the sensor's detected values, the winner will not be rewarded. We had to make it before implementing Richard's approach, because the winners were getting their strength growing apparently very fast. It looks as not being required any more, as some trials had shown. The third is because Richards [8] introduces a new classifier when it doesn't occur matching, and this classifier has its antecedent equal to the sensors codified values, and the consequent is generated randomly. As we will see in the next section, the specificity is in CSNN a rather abstract concept. Anyway, to generate a Neural Network with a very high specificity, related to the other classifiers, is not a straightforward procedure. But this new classifier represents an automated way of putting knowledge inside the system, and certainly we will incorporate a similar procedure in our algorithm.

4) The controller: a MATLAB script is used as a controller for the process, as presented in Figure 6. *Parameter initialization* is related to all required parameters for configuring the type of experimentation being held. *Environment definition* creates the figure that will bear the environment with its objects, and defines the environment contour. *Objects definitions* settle all the objects that will be used. Of course, an object can be settled at any time, not just at this point. *RobotMove definition* will settle the robots in the environment. *Classifier initialization* will generate the initial set of classifiers, usually by means of a random process. With the actuator, the main cycle is started: it calls the method *Actuator* to obtain the control variables to the robot; in the first passage, the actuator is useless, since nothing happened yet. The location of the robot will be based on the outputs obtained by the neural networks of the *Actuator* method. Afterwards the method *set* for the *RobotMove* will settle its new data and plot in the new position. It's also useless in the first passage of the main cycle. Next, we have to check if the action was successful by verifying the absence of a collision and getting the new sensors readings. All of this is done calling the method *Detect*. *PunishmentReward* will evaluate the action proposed by the winner, beginning at the second passage. The Genetic Algorithm is triggered by one of two occurrences, whichever comes first: a given number of collisions and a given number of cycles since the last time it was called. Calling *Classification* we obtain the classification, using as input the sensors data got by *Detect*. And so we start a new iteration with the system going back to the *Actuator* step.

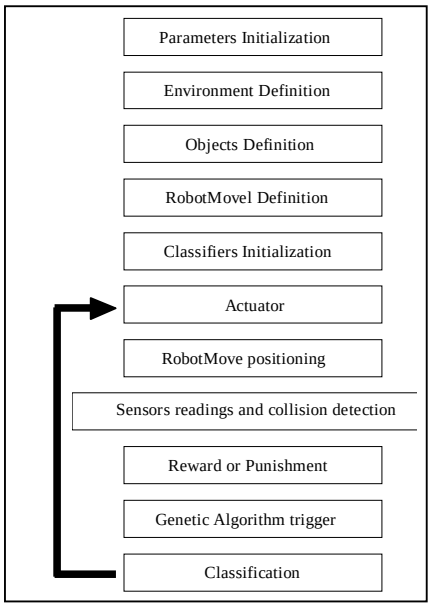

Figure 6 – Steps of the controller

IV. NEURAL NETWORKS AS CLASSIFIERS: THE MAIN CONSEQUENCES

The introduction of neural networks to replace the rule base classifiers in classifiers systems modifies its behavior in ways that must be understood to be able to explore its potentials and/or even to allow the increment in performance. Here we will talk about some of these differences that we have detected.

A. Matching

In conventional classifier systems (CS), matching is a distinct occurrence. It happens or not. In classifier system with neural networks (CSNN), matching is a real number, an output of the Evaluation Neural Network. So we'll have to specify the matching occurrence by choosing the range of values that will define it. At the beginning, we considered that all classifiers could match any message, and that the output of the evaluation network would define the classifier specificity. This fact allows weak classifiers to win the auction, when its specificity turned to be negative, while its strength had become negative. The product of them produces a positive number to formulate the classifier bid. So we established that only positive output determines the matching. For simplicity, we still considered the same number for matching and specificity, but we think that it will be worth to introduce one more neuron in the output layer of the network to define the classifier specificity.

B. Specificity

In CS, specificity is a characteristic of the classifier. In CSNN it is an output of the evaluation network, having as input the codification of the data provided by the sensors. It means that the same classifier may have different specificity for different messages. That is, specificity in CSNN is not a characteristic of the classifier. It represents a generalized concept, meaning that a classifier will be more specific for a pattern of messages and less specific for other, expressing a more complex behavior.

C. Consequent

In CS, a classifier can be associated with different messages, depending on its specificity. The same occurs in CSNN, but in a rather sophisticated manner, as we saw in the items above. Now, looking at the action that this classifier will produce, we see that in conventional CS it is always the same for all the messages it attends, because the action is related to its consequent. However, for the CSNN it is a function of inputs, as the consequent represents an action network, with different input for different messages.

D. Number of classifiers required to solve a stationary environment

In the CS this number corresponds to the number of required situations for the command of the actuators, because even having general classifiers, their consequent will give the same output for the different messages. One case that can be solved by just one classifier would happen in a circular movement in a non-symmetric region. In this movement the

sensors will produce different readings that will have to match a general classifier with enough strength to compensate its low specificity. This classifier will produce always the same output (for instance - turn right 10 degrees) in accordance with its consequent.

In the CSNN, it is theoretically possible for just one classifier to solve a more complex then a circular movement, requiring different outputs from the action network. In one way this possibility points to a promising feature. Consider the fact that neural networks are universal approximators [5]. It sounds reasonable to expect a better solution then the obtained by the CS. But we have to consider that, in this case, we will have only one winner, diminishing the significance of the credit assignment process, once the networks achieve the required training. We think that it would be better to generate the solution with more classifiers participating, even if their number remain lesser then the required by conventional CS. Notice that we are dealing with no stationary environments and we have to maintain the strength of the classifiers.

E. Initial knowledge and external knowledge injection

No doubt that autonomous robots utilizing learning techniques demand less design effort when compared to expert systems, where all the rules should be defined by someone, normally an expert. But it is a good practice to reduce the learning effort introducing some available knowledge in the initialization and even during the training, mostly when this activity is not very difficult. In conventional CS, it is easier then in the CSNN because in that system the mapping from sensors data to antecedent + consequent is direct. So, initialize the classifiers with external knowledge or even introduce knowledge as suggested by Richards [8] when there is no matching is always easier in the CS.

V. SOME PRELIMINARY RESULTS AND CONCLUSIONS

We have just finished the software and have obtained some preliminary results. In the experiments, the robot has a fixed speed, and modifying one of its wheels speed changes its orientation. It has to run avoiding obstacles.

There is a noticeable dependency on the initial conditions. There are some ways of minimizing it, as increasing the number of classifiers, at the cost of increasing the computational effort. As next steps, we intend to introduce knowledge automatically and to use reinforcement learning [1].

Some experiments showed that just one classifier could handle simple environment demands. The robot converges to a circular-like movement, in a non-symmetric region of the environment. Now we are going to investigate whether this possibility is consequence of the use of neural networks or is caused by the balance of reward and punishment we are using.

In some cases we had two classifiers working after the learning phase, and the robot presents a trajectory that covers all the environment area. We are going to contrast this case with the one cited above. We are also devising ways to check the quality of the learned behavior, maybe including more obstacles or adopting a non-stationary environment.

In some non-succeeded experiments, we observed a behavior in which the robot was caught in a kind of trap. It indefinitely stayed repeating the same sequence of steps towards a collision. And we noticed few classifiers being responsible for that repeated action. The treatment of this behavior will probably involve the evaluation of the classifier that determines the collision. If it is a generic classifier, then it is possible that in some steps it gets rewards because there is no collision and in just one step, when the collision occurs, it is punished. This generic classifier with bad and good responses is not being rejected by the credit assignment and will require a specific treatment.

As we mentioned before, the results of this paper is part of a broader project, and the preliminary results described here may indicate future steps. All of our efforts are intended to provide useful insights in order to abbreviate the implementation of real-world autonomous navigation based on learning evolutionary algorithms.

VI. BIBLIOGRAFY

[1] Ackley, D., and Littman, M. “Interactions Between Learning and Evolution”, *Artificial Life II*, SFI Studies in the Sciences of Complexity, vol. X, edited by C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen, Addison-Wesley, 1991.

[2] Booker, L. B., Goldberg, D. E., and Holland, J. H. “Classifier Systems and Genetic Algorithms”, Artificial Intelligence, 40: 235-282, 1989.

[3] Goldberg, D.E. “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley, 1989.

[4] Holland, J. H. “Adaptation in Natural and Artificial Systems”, University of Michigan Press, An Arbor, MI, 1975.

[5] Hornik, K., Stinchcombe, M., and White, H. “Multi-layer feedforward networks are universal approximators”, *Neural Networks*, 2(5): 359-366, 1989.

[6] Moussi, L.N., Gudwin, R.R., Von Zuben, F.J., Madrid, M.K. Neural networks in classifier systems (NNCS): An application to autonomous navigation. *in* V.V. Kluev & N.E. Mastorakis (eds.) *Advances in Signal Processing, Robotics and Communications*, Electrical and Computer Engineering Series, WSES Press, pp. 256-262, 2001.

[7] Ramsey, C.L., Schultz, A.C., and Grefenstette, J.J. “Simulation-assisted learning by competition: Effects of noise differences between training model and target environment”, Proceedings of the Seventh International Conference on Machine Learnig, Austin, TX, Morgan Kaufmann , pp. 211-215, 1990.

[8] Richards, R.A. “Zeroth-Order Shape Optimization Utilizing a Learning Classifier System”, Ph.D. Thesis, Mechanical Eng. Dept., Stanford University, 1995. http://www.stanford.edu/~buc/SPHINcsX/book.html

[9] Schultz, A.C., and Grefenstette, J.J. “Using a Genetic Algorithm to Learn Behaviors for Autonomous Vehicles”, Navy Center for Applied Research in Artificial Intelligence, Navy Research Laboratory, Washington, DC, *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Hilton Head, SC, August 10-12, 1992.

[10] Webots Release 3.0.1 (for evaluation purpose only), from Cyberbotics Ltd., www.cyberbotics.com.

ACKNOWLEDGMENTS: The authors acknowledge FAPESP, the research agency for the state of São Paulo, CNPq, the Brazilian National Research Council (grants 300910/96-7 and 300123/99-0) and CAPES for their support to this work.