

# Virtual Environment Using Cellular Automata for Experimental Studies of Sound Propagation and Echo Classification with Artificial Neural Networks

M. Bucurica, R. Dogaru, *Member, IEEE*

**Abstract**—This work presents a Java platform capable to emulate sound propagation in a controlled 2D environment (obstacles and sound sources selected by the user) based on a cellular automata model. The platform is expandable and so far includes a feature preprocessor for the echo waves and a neural classifier. The proposed virtual environment allows performing various virtual experiments with relevance in robotics and Simultaneous Localization and Mapping (SLAM). Performing such virtual experiments allows important reduction in development cost and the fine tuning of a solution for a given problem before testing it on a real ultrasound workbench.

**Index Terms**—Cellular neural networks, Acoustic waves, Neural networks, Radial basis function networks.

## I. INTRODUCTION

Using ultrasound transmitters and sensors is a convenient and computationally effective method for an autonomous agent (e.g. robot) to learn about its environment. Various problems occur in its interaction with the environment, starting from obstacle avoidance to more complex problems like creating an internal map of the obstacles to be further used. Performing such experiments in a virtual environment before designing the real hardware objects allows to reduce development costs.

Herein we present a JAVA platform implementing such a virtual environment, which adds to a wave propagation model based on cellular automata several computational intelligence features to perform various virtual experiments corresponding to the intelligence of robots aiming to understand their environments from echo signals in an attempt to mimic bats and other kind of living creatures [1]. Although other sound propagation simulators were previously reported in the literature [2] to our knowledge, no other such simulators include intelligent echo processing functions. Section II reminds the cellular automata model used [3] to model sound

propagation. Our main contribution in developing the Java platform with artificial intelligence tools to interpret signals from the environment are presented in Section III and some relevant experimental results are given in Section IV.

## II. THE CELLULAR AUTOMATA VIRTUAL SCENARIO KERNEL

The kernel of the virtual environment is based on CANAVI [3], where cellular automata are used as a relatively accurate model of wave propagation in an arbitrary given 2-dimensional scenario. Starting from the one-dimensional wave equation in [4] a cellular automata model was first proposed, further refined by us in [5] as a 2-dimensional virtual space of NxM cellular automata cells (detailed in Fig. 1).

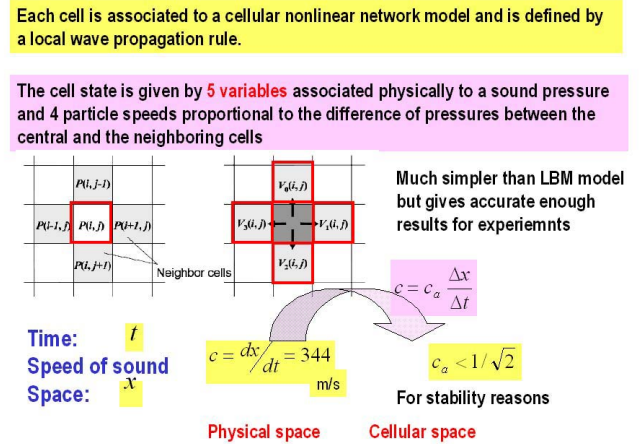


Fig.1 Cell structure and neighbors in the CANAVI model [5]

Each CA cell was initially associated to 5 state variables but later in [4] a speed-up was obtained reducing each cell to only 2 state variables,  $P$  and  $SV$  updated synchronously at each iteration corresponding to the physical space.

$$P(t) = S(t) \quad (1)$$

if the cell is a source signal

$$P(t) = P(t-1) - c_a^2 SV(t-1) \quad (2)$$

if otherwise (obstacle, air, receiver)

M. Bucurica is with the Doctoral School on Electronics Telecommunications and Information Technology (e-mail: bucurica.mihai@gmail.com)

R. Dogaru is with the Natural Computing Lab, Dept. of Applied Electronics and Information Engineering, University "Politehnica" of Bucharest (e-mail: radu\_d@ieee.org).

where

- $P(t)$  is the pressure of the cell,
- $P(t-1)$  is the pressure of the cell at the previous moment of time,
- $SV(t-1)$  is the cumulated velocity of the neighbor cells as described below in (3),
- $c$  is a constant parameter called the “virtual speed of sound”

$$SV = [1 - d(i, j)]SV^- + 4P - (P_L + P_R + P_U + P_D) \quad (3)$$

where

- $SV$  is the current computed pressure
- $SV^-$  is the computed pressure of the neighbor cells from the previous moment of time
- $d(i, j)$  is the absorbance parameter of the material previously optimized by comparing real with virtual experiments:
  - for air is : 0.01
  - for any other obstacle is : 0.8
- $P$  is the previously updated (1)(2) pressure of the cell
- $P_L, P_R, P_U, P_D$  are the previously updated pressures from the neighbor cells.
- $i, j$  are the cell spatial coordinates.

The link between physical space and virtual space is given by the equation  $c = c_a(\Delta x / \Delta t)$ , where  $c_a$  is the constant parameter called the “virtual speed of sound” used in the (2), usually chosen as 0.688 while  $c$  is the speed of sound (344 m/s at normal pressure and temperature). For instance, choosing a sampling frequency of 40 kHz results that each cell will model a fraction  $\Delta x = 0.0125m$ .

Any arbitrary scenario may be created allowing that each of the CA cell may be in certain categories: i) it can be a part of a pressure generator (transmitter) associated with a (ultra)sonic signal source (sampled with period  $\Delta t$ ); ii) it can be simply air (or other transmission medium); iii) it can be part of an obstacle. Generally, the application uses the principle of echolocation: an short ultrasound pulse is emitted (for the firsts  $T = 20 - 80$  iterations) and then the receiver waits for the echoes.

The simulation will run for  $T$  iterations. Generally the  $T$  used in our experiments ranges from 1000 to 1200. The mean execution time for a virtual space of  $150 \times 150$ , (i.e. 22500 cells) is around 4.6 seconds corresponding to a mean execution time per cell of about  $150 - 160 \times 10^{-9}$  seconds (150-160 nanoseconds).

The Java virtual environment allows the editing of various scenarios and then display the sound wave dynamics, as shown in Fig. 2.

### III. ADDING INTELLIGENCE TO OUR VIRTUAL ENVIRONMENT

Compared to the previous implementations [3][5] (developed in Octave/C++) this Java platform is a completely new implementation which adds in functionality the artificial intelligence modules, not present in the previous approaches.

Their role is mainly to interpret signals received at arbitrary points (echoes) for various tasks. Such computational intelligence modules will be later included as parts of the autonomous agents (robots) to perform their particular tasks.

In terms of the implementation accuracy, similar wave propagation scenarios were tested in both the platform in [3][5] and the actual Java platform concluding that no errors are present between simulations (the same CA model is used in both platforms).

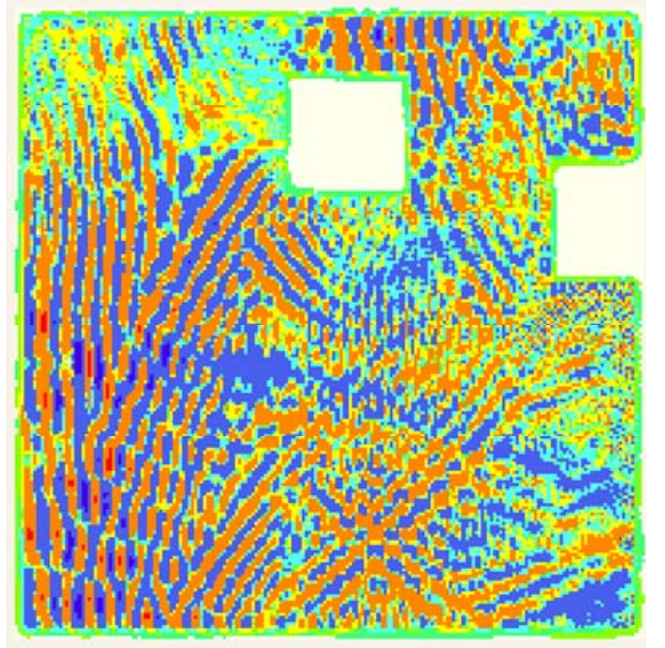


Fig. 2. Image representing a sound wave simulation run with the software. The white spaces represents obstacles/objects and rest of the colors represents the amplitude of the sound waves (negative and positive)

So far, these computational intelligence features include the following: i) A feature vector processing unit (FVPU) which takes an arbitrary long echo and transforms it into a fixed size vector acting as input to a neural network; ii) A database generator – which generates collections of feature vectors and desired outputs, specific to a certain kind of experiment; iii) A neural network capable to be trained for the generated databases.

The FVPU is inspired from neuro-fuzzy processing and divides the entire length of the echo signal into a certain number  $m = 2nq + 1$  of overlapping regions defined by triangular windows. Each triangular window corresponds to one element of the  $m$ -dimensional feature vector that is computed as the energy of the triangularly weighted signal within the particular window.

Usually in the application the effective number of triangles is  $m = 2nq - 1$ , because the first and the last triangle are never used because of their irrelevant information.

The choice of the  $nq$  parameter is a matter of an optimization process aiming to get the best performance for a given experiment. More details will be given in Section IV.

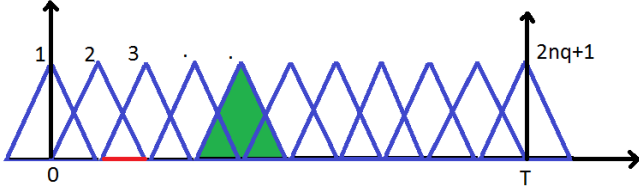


Fig. 3. Image representing the feature extractor within a window of the echo signal

The neural network used herein is the RBF-M (or recently renamed Fast Support Vector Classifier) previously introduced in a series of papers [6][7]. It combines the guaranteed convergence learning of the output Adaline layer with a non-linear pre-processor made of a number of RBF units, thus providing the work of Adaline in an expanded space and consequently being capable to represent complex data models. It has the advantage of simple training with an algorithm that automatically constructs the number of RBF units given a unique radius parameter. This parameter is usually the only one which has to be optimized, although for a fine tuning the second parameter (threshold, or overlapping factor  $ov$ ) can be also considered.

Creation of the cells (RBF units):

- the first training vector is selected as the first neuron.
- threshold = 1 is set for creating a new neuron.
- the rest of the feature vectors from the training set are parsed in one epoch and for each new one an activity number is computed which if is lower than the threshold a new neuron is created with a *center vector* copying the actual feature vector. The activity is computed as a sum of all existing RBF units for the actual feature vector.
- the maximal number of RBF units equals the number of samples in the training set, although by varying the radius parameter one will find the best trade-off corresponding to the optimal number of RBF units.

Training the neural network:

- the RBF neurons are created with the above algorithm
- each weight is initialized with 0
- training and testing error are initialized with a high value : 9999
- for a predefined number of epochs (in our experiments: 200), cross-validation is performed and the best (smallest error) result on the test set is saved as well as its corresponding structure (RBF center vectors and Adaline weights).
- finding the lowest error value, multiple tests are run with radius value between 0.1 and 5
- after finding the lowest testing error value, the weights saved in the row vector are the optimal weights and will be used in future testing.

Depending on the experiment, databases are created to allow training of the neural network to classify the obstacle (the object), to determine their shape, or to estimate the distance to the closest object, etc.

An example of training the neural network to estimate the location of the agent based on the received echoes is given in Fig. 4. It corresponds to the best result obtained for such a difficult task.

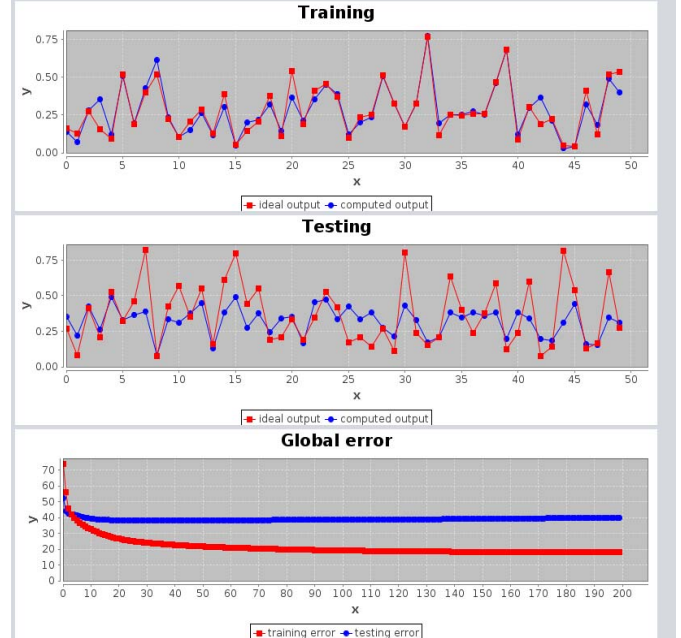


Fig. 4. Snapshot of the Java simulator during a training/test procedure

#### IV. EXPERIMENTAL RESULTS

The tests we conducted were separated in two parts: one where we tried to determine if the software for the platform will be fast enough to be considered as an alternative and cheap solution to the existing software – *performance of the CA* and another one where we tried to determine if the principle of the echolocation can be used to determine the distance to the nearest obstacle from the receiver inside the virtual space of simulation – *performance of the neural network and of the feature extractor*.

The system we run the tests on is a personal computer clocked at 2.26 GHz (Intel Pentium P8400) with 6GB of memory at speed of 1066 MHz (DDR3). Each simulation ran for  $T=1200$  clock iteration, and for each test we noted the execution time (Et) in seconds and the execution time per cell (Etpc) in nanoseconds.

To be noted that the state of the processor during the tests was between 90 -100% load.

##### A. Performance of the CA:

**Scenario 1:** Virtual room space:  $150 \times 150 = 22\,500$  cells – plotting the scenario : ON

	1	2	3	4	5
Et(s)	4.602	4.540	4.524	4.555	4.384
Etpc(ns)	170.44	168.14	167.55	168.70	162.37

The average time of Etpc is about  $167 \times 10^{-9}$  seconds.

**Scenario 2:** Virtual room space: 200 x 200 = 40 000 cells – plotting the scenario: OFF

	1	2	3	4	5
Et(s)	7.242	7.242	7.209	7.230	7.194
Etpc(ns)	150.87	150.87	150.20	150.54	149.89

The average time of Etpc is about  $150 \cdot 10^{-9}$  seconds, a slight increased performance.

*B. Comparison between the previous implementation[3][5] with the current one of the CA.*

**Virtual room space:** 200 x 200 = 40 000 cells

	OCTAVE/C++		JAVA	
	Et(s)	Etpc(ns)	Et(s)	Etpc(ns)
1	2.386	49.232	7.242	150.87
2	2.376	49.232	7.242	150.87
3	2.318	47.829	7.209	150.20
4	2.332	48.109	7.230	150.54
5	2.320	47.862	7.194	149.89

It can be seen that the average time of Etpc which translate in the performance per cell is about 150 nanoseconds when using the JAVA platform compared with the 49 nanoseconds when using the Octave/C++ implementation. In the future research, one goal is to decrease the Etpc to at least the score in C++ platform, by using parallel processing and/or multithreading.

*C. Performance of the neural network and of the feature extractor.*

**Scenario 1:**

- feature vector parameter  $m=39$
- database sets of training and testing : 50 samples

Radius	1	2	3
training error	0.007%	4.65%	12.48%
testing error	46.15%	32.84%	41.84%
RBF units	25	23	18

**Scenario 2:**

- feature vector parameter  $m=39$
- database sets of training and testing : 100 samples

Radius	1	1.5	2	2.1	2.5
training error	0.002%	0.10%	10.09%	8.67%	5.37%
testing error	82.01%	55.59%	27.58%	27.80%	32.64%
RBF units	50	49	43	43	37

**Scenario 3:**

- feature vector parameter  $m=39$
- database sets of training and testing : 200 samples
- 

Radius	1.5	2	2.1	2.2	2.4	2.5
training error	3.30 %	11.83 %	15.87 %	16.81 %	25.18 %	24.09 %
testing error	53.1 %	43.00 %	39.54 %	38.08 %	32.09 %	35.45 %
RBF units	86	67	64	60	50	48

## V. CONCLUSIONS

The best error value was 27.58%, obtained for a number of 100 samples and  $m=39$ . This error represents the difference in length between the two distances: the real distance and the distance estimated by the neural network based on the received echoes.

The presented platform is developed as an extensible one, starting from a CA-based sound propagation model as a kernel. As demonstrated in this paper, computational intelligence tools are added (not present in previous similar simulators ([3][5]) as useful tools to be applied in various experiments that are relevant in various practical circumstances. Herein we presented some examples in learning a relationship between echoes and the position of the robot relative to the closest obstacle but many other experiments may be performed using the same tools, as we plan for our further research, for instance the classification of objects based on received echoes.

## REFERENCES

- [1] Nobuo Suga, "Biosonar and Neural Computation in Bats", Scientific American 60-68, June 1990.
- [2] E.M. Viggen, "The Lattice Boltzmann Method with Applications in Acoustics", Master thesis, Department of Physics - NTNU 2009
- [3] R. Dogaru and I. Dogaru, "An Efficient Sound Propagation Software Simulator Based on Cellular Automata", in Proceedings ISEEE 2010 (September 2010, Galati, Romania), pp. 273-276.
- [4] T. Komatsuzaki and Y. Iwata, "Study on Acoustic Field with Fractal Boundary using Cellular Automata", *Proc. of ACRI2008* Springer LNCS5191, pp. 282-290.
- [5] Radu Dogaru, Ioana Dogaru, "Efficient Realizations of a Sound Propagation Processor as a Cellular Nonlinear Network", in Proceedings of 10th International Symposium on Signals, Circuits and Systems (ISSCS), 2011, June 30 2011-July 1, 2011, Iasi, Romania, pp. 165-168.
- [6] Dogaru R (2007) A hardware oriented classifier with simple constructive training based on support vectors , in Proceedings of CSCS-16, the 16th International Conference on Control Systems and Computer Science, May 22 - 26, 2007, Bucharest, Romania, Vol.1, pp. 415-418, ISBN 978-973-718-741-3.
- [7] R. Dogaru and I. Dogaru, "An Efficient Finite Precision RBF-M Neural Network Architecture Using Support Vectors", in Proceedings 10's Symposium on Neural Networks Applications in Electrical Engineering, NEUREL2010, Sept. 23-25, 2010, pp. 127-130.