Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2019, 6(1):32-39

Research Article ISSN: 2394 - 658X

Autonomous Driving using Stochastic Learning Automata

Florin Stoica

Department of Mathematics and Informatics, "Lucian Blaga" University of Sibiu, Sibiu, Romania
florin.stoica@ulbsibiu.ro

ABSTRACT
An automaton is a machine or control mechanism designed to execute a default sequence of operations or to respond to instructions. The term stochastic captures the adaptive nature of the automata described in this paper. Such an automaton does not follow predefined rules, but adapts to its environment. This adaptation is the result of a learning process. Through learning, is understood any permanent change in behavior as a result of past experience. For this reason, a learning system must have the ability to improve its behavior over time to achieve a final goal. A stochastic machine tries to find a solution to the problem without any information about the optimal action. The learning process is implemented through a reinforcement algorithm. The reinforcement scheme presented in this paper is shown to satisfy all necessary and sufficient conditions for absolute expediency for a stationary environment. An automaton using this scheme is guaranteed to „do better" at every time step than at the previous step. Using Stochastic Learning Automata, we introduce a decision/control method for intelligent vehicles to simulate autonomous driving. A multi-agent approach is used for effective implementation. Each vehicle has associated a "driver" agent, hosted on a JADE platform.

Key words: Stochastic Learning Automata, Reinforcement Learning, Intelligent Vehicle Control, agents

INTRODUCTION

The operation of a stochastic machine is the following: an action is selected from the set of actions that the machine can execute based on the probabilities associated with these actions, then the environmental response is observed after the execution of the action and finally the probabilities associated with actions are eventually updated, repeating the procedure. A stochastic automaton acting in the manner described in order to improve its performance is called a learning automaton. The algorithm that guarantees the desired learning process is called a reinforcement scheme [1].

Mathematically, the environment is defined by a triple $\{\alpha, c, \beta\}$ where $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ represents a finite set of actions being the input to the environment, $\beta = \{0,1\}$ represents a binary response set and $c = \{c_1, c_2, ..., c_r\}$ is a set of penalty probabilities, where c_i is the probability that executing the action α_i will result in an unfavorable response from the environment. Given that $\beta(n) = 0$ is a favorable outcome and $\beta(n) = 1$ is an unfavorable outcome at time instant n $(n = 0, 1, 2, ...)$, the element c_i of c is defined mathematically by: $c_i = P(\beta(n) = 1 | \alpha(n) = \alpha_i)$ $i = 1, 2, ..., r$.

The environment can be split up in two types, stationary and nonstationary. In a stationary environment the penalty probabilities will never change. In a nonstationary environment the penalties will change over time.

FORMAL DEFINITION OF A STOCHASTIC LEARNING AUTOMATON

The automaton is defined by a quintuple $\{\Phi, \alpha, \beta, f, g\}$.

$\Phi = \{\Phi_1, \Phi_2, ..., \Phi_s\}$ is the set of internal states of the automaton. The internal states determine the action to be executed by the automaton. The state at time instant n is denoted by $\Phi(n)$ and is an element of the finite set Φ.

$\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ denotes the set of actions that can be executed by the automaton. This is the output set of the automaton, hence also being the input set to the environment. The action done at time instant n is denoted $\alpha(n)$ and is an element of the finite set α.

$\beta = \{0,1\}$ is the input set to the automaton, that is the set of responses from the environment. $\beta(n)$ denotes the input to the automaton at time instant n.

The elements f_{ij}^{β} of transition function f represent the probability that the automaton moves from state Φ_i to state Φ_j given the input β:

$$f_{ij}^{\beta} = P(\Phi(n+1) = \Phi_j \mid \Phi(n) = \Phi_i, \beta(n) = \beta)\ i,\ j = 1, 2, ..., s \text{ and } \beta \in \{0,1\}$$

The elements of the output function g are denoted g_{ij} and represent the probability that the action done by the automaton is α_j given the automaton is in state Φ_i:

$$g_{ij} = P(\alpha(n) = \alpha_j \mid \Phi(n) = \Phi_i)\ \ i = \overline{1,s}\ \ j = \overline{1,r}$$

By making the values f_{ij}^{β} and g_{ij} change over time, the automaton has greater flexibility, allowing the rewarded actions to get a higher chance of being chosen again. Such an automaton is called *variable-structure automaton*. In order to describe a reinforcement scheme for such an automaton, is defined $p(n)$, an array of action probabilities:

$$p_i(n) = P(\alpha(n) = \alpha_i),\ \ i = \overline{1,r}$$

Updating action probabilities can be represented as follows:

$$p(n+1) = T[p(n), \alpha(n), \beta(n)]$$

where$T$ is a mapping. This formula says the next values of action probabilities $p(n+1)$ are calculated based on the current values of action probabilities $p(n)$, the chosen action and the input from the environment.

STOCHASTIC LEARNING AUTOMATON WITH VARIABLE STRUCTURE

Performance Evaluation

A learning automaton must have the ability to improve its behaviour over time to achieve a final goal and its performance must be superior to "intuitive" method.

Consider a stationary environment with penalty probabilities $\{c_1, c_2, ..., c_r\}$ where $c_i = P(\beta(n) = 1 \mid \alpha(n) = \alpha_i)$. We define a quantity $M(n)$ as the average penalty for a given action probability array:

$$M(n) = P(\beta(n) = 1 \mid p(n)) = \sum_{i=1}^{r} P(\beta(n) = 1 \mid \alpha(n) = \alpha_i) * P(\alpha(n) = \alpha_i) = \sum_{i=1}^{r} c_i p_i(n)$$

An automaton is absolutely expedient if the expected value of the average penalty at one iteration step is less than it was at the previous step *for all steps*: $M(n+1) < M(n)$ for all n [2].

Absolutely expedient learning schemes are presently the only class of schemes for which necessary and sufficient conditions of design are available. The general solution for absolutely expedient schemes was found by Lakshmivarahan and Thathachar [3].

The characterization theorem of absolutely expedient reinforcement schemes

The reinforcement scheme is the basis of the learning process for stochastic learning automata. A learning automaton may send its action to multiple environments at the same time. In that case, the action of the automaton results in an array of responses from environments. In a stationary model with N environments, if an automaton executed the action α_i and the environments responses are $\beta_i^j\ j = 1, ..., N$ at time instant n, then the vector of action probabilities $p(n)$ is updated as follows [4]:

$$p_i(n+1) = p_i(n) + \left[\frac{1}{N}\sum_{k=1}^{N}\beta_i^k\right] * \sum_{\substack{j=1 \\ j \neq i}}^{r} \varphi_j(p(n)) - \left[1 - \frac{1}{N}\sum_{k=1}^{N}\beta_i^k\right] * \sum_{\substack{j=1 \\ j \neq i}}^{r} \psi_j(p(n))$$

$$p_j(n+1) = p_j(n) - \left[\frac{1}{N}\sum_{k=1}^{N}\beta_i^k\right] * \varphi_j(p(n)) + \left[1 - \frac{1}{N}\sum_{k=1}^{N}\beta_i^k\right] * \psi_j(p(n)) \quad (1)$$

for all $j \neq i$ where the functions φ_i and ψ_i satisfy the following conditions:

$$\frac{\varphi_1(p(n))}{p_1(n)} = ... = \frac{\varphi_r(p(n))}{p_r(n)} = \lambda(p(n)) \quad (2)$$

$$\frac{\psi_1(p(n))}{p_1(n)} = ... = \frac{\psi_r(p(n))}{p_r(n)} = \mu(p(n))$$

$$p_i(n) + \sum_{\substack{j=1 \\ j \neq i}}^{r} \varphi_j(p(n)) > 0 \quad (3)$$

$$p_i(n) - \sum_{\substack{j=1 \\ j \neq i}}^{r} \psi_j(p(n)) < 1 \tag{4}$$

$$p_j(n) + \psi_j(p(n)) > 0 \tag{5}$$

$$p_j(n) - \varphi_j(p(n)) < 1 \tag{6}$$

for all $j \in \{1,...,r\} \setminus \{i\}$

The conditions (3)-(6) ensure that $0 < p_k < 1,\ k = \overline{1,r}$.

Theorem If the functions $\lambda(p(n))$ and $\mu(p(n))$ satisfy the following conditions:

$$\begin{aligned} &\lambda(p(n)) \le 0 \\ &\mu(p(n)) \le 0 \\ &\lambda(p(n)) + \mu(p(n)) < 0 \end{aligned} \tag{7}$$

then the automaton with the reinforcement scheme defined by (1) is absolutely expedient in a stationary environment. The proof of this theorem can be found in [5].

A NEW ABSOLUTELY EXPEDIENT REINFORCEMENT SCHEME

Because the above theorem is also valid for a single-environment model, we can define a single environment response that is a function f of many environments outputs.

Thus, we can update the above algorithm as follows:

$$\begin{aligned} &p_i(n+1) = p_i(n) + f * (-\delta * H(n)) * [1 - p_i(n)] - (1-f) * (-\theta) * [1 - p_i(n)] \\ &p_j(n+1) = p_j(n) - f * (-\delta * H(n)) * p_j(n) + (1-f) * (-\theta) * p_j(n) \end{aligned} \tag{8}$$

for all $j \neq i$, i.e.:

$$\psi_k(p(n)) = -\theta * p_k(n)$$

$$\varphi_k(p(n)) = -\delta * H(n) * p_k(n)$$

where the learning parameters θ, δ are real values which satisfy: $0 < \theta < 1, 0 < \delta < 1$ and $0 < \theta + \delta < 1$.

The function H is defined as:

$$H(n) = \min\left\{1; \max\left\{\min\left\{\frac{p_i(n)}{\delta(1-p_i(n))} - \varepsilon, \left(\frac{1-p_j(n)}{\delta * p_j(n)} - \varepsilon\right)_{\substack{j=\overline{1,r} \\ j \neq i}}\right\}; 0\right\}\right\}$$

Parameter ε is an arbitrarily small positive real number.

This reinforcement scheme differs from the one given in [4], [6] by the definition of these two functions: H and ϕ_k .

We now show that are satisfied all the conditions of the reinforcement scheme.

From (2) we have:

$$\frac{\varphi_k(p(n))}{p_k(n)} = \frac{-\delta * H(n) * p_k(n)}{p_k(n)} = -\delta * H(n) = \lambda(p(n))$$

$$\frac{\psi_k(p(n))}{p_k(n)} = \frac{-\theta * p_k(n)}{p_k(n)} = -\theta = \mu(p(n))$$

The rest of the conditions translate to the following:

Condition (3):

$$p_i(n) + \sum_{\substack{j=1 \\ j \neq i}}^{r} \varphi_j(p(n)) > 0 \Leftrightarrow p_i(n) - \delta * H(n) * (1 - p_i(n)) > 0 \Leftrightarrow \delta * H(n) * (1 - p_i(n)) < p_i(n) \Leftrightarrow H(n) < \frac{p_i(n)}{\delta * (1 - p_i(n))}$$

This condition is satisfied by the definition of the function $H(n)$.

Condition (4): $p_i(n) - \sum_{\substack{j=1 \\ j \neq i}}^{r} \psi_j(p(n)) < 1 \Leftrightarrow p_i(n) + \theta * (1 - p_i(n)) < 1$

But $p_i(n) + \theta * (1 - p_i(n)) < p_i(n) + 1 - p_i(n) = 1$ since $0 < \theta < 1$

Condition (5):

$p_j(n) + \psi_j(p(n)) > 0 \Leftrightarrow p_j(n) - \theta * p_j(n) > 0$ for all $j \in \{1,...,r\} \setminus \{i\}$

But $p_j(n) - \theta * p_j(n) = p_j(n) * (1 - \theta) > 0$ since $0 < \theta < 1$ and $0 < p_j(n) < 1$ for all $j \in \{1,...,r\} \setminus \{i\}$

Condition (6): $p_j(n) - \varphi_j(p(n)) < 1 \Leftrightarrow p_j(n) + \delta * H(n) * p_j(n) < 1$ for all $j \in \{1,...,r\} \setminus \{i\}$

We have:

$$p_j(n) + \delta * H(n) * p_j(n) < 1 \Leftrightarrow H(n) < \frac{1 - p_j(n)}{\delta * p_j(n)} \text{ for all } j \in \{1,...,r\} \setminus \{i\}.$$

This condition is satisfied by the definition of the function $H(n)$.

With all conditions of the equations (1) satisfied, we conclude that the reinforcement scheme is a candidate for absolute expediency.

Furthermore, the functions λ and μ for our nonlinear scheme satisfy the following:

$$\lambda(p(n)) = -\delta * H(n) \leq 0$$

$$\mu(p(n)) = -\theta \leq 0$$

$$\lambda(p(n)) + \mu(p(n)) = -\delta * H(n) - (1 - \delta) = -\delta * H(n) - 1 + \delta = \delta(1 - H(n)) - 1$$

Then $\lambda(p(n)) + \mu(p(n)) < 0 \Leftrightarrow \delta(1 - H(n)) - 1 < 0 \Leftrightarrow \delta(1 - H(n)) < 1$, which is true because $0 < \delta < 1$ and $0 \leq H(n) \leq 1$.

In conclusion, we state the algorithm given in equations (8) is absolutely expedient in a stationary environment.

AUTONOMOUS DRIVING USING STOCHASTIC LEARNING AUTOMATA

Reinforcement learning is justified if the behavior generated by an agent presents desirable emergent properties (like generalization, robustness, redundancy, adaptability) which cannot be directly built. This reason represents a good motivation for the use of reinforcement learning in intelligent vehicle control [7].

In this section, we present a method for intelligent vehicle control, having as theoretical background Stochastic Learning Automata. We attempt to find a way to make intelligent decisions here, having as objectives conformance with traffic parameters imposed by the user, and improving safety by minimizing crash risk.

The aim here is to design an automata system that can learn the best possible action based on the data received from on-board sensors. For our model, we assume that an intelligent vehicle is capable of two sets of lateral and longitudinal actions. Lateral actions are LEFT (shift to left lane), RIGHT (shift to right lane) and LINE_OK (stay in current lane). Longitudinal actions are ACC (accelerate), DEC (decelerate) and SPEED_OK (keep current speed).

An autonomous vehicle must be able to “sense” the environment around itself. Therefore, we assume that there are four different sensors modules on board the vehicle (the frontal module, two side modules and a speed module), in order to detect the presence of a vehicle traveling in front of the vehicle or in the immediately adjacent lane and to know the current speed of the vehicle.

The response from physical environment is a combination of outputs from the sensor modules. Because an input parameter for the decision blocks is the action chosen by the stochastic automaton, is necessary to use two distinct learning automata, namely the longitudinal automaton and respectively the lateral automaton.

After updating the action probability arrays in both learning automata, using the new reinforcement scheme presented in this paper, the outputs from stochastic automata are transmitted to the regulation layer. The regulation layer handles the actions received from the two automata in a distinct manner, using for each of them a regulation buffer. If an action received was rewarded, it will be introduced in the regulation buffer of the corresponding automaton, else in buffer will be introduced a certain value which denotes a penalized action by the physical environment. The regulation layer does not carry out the action chosen immediately; instead, it carries out an action only if it is recommended k times consecutively by the automaton, where k is the length of the regulation buffer. After executing an action, the probability array is initialized so that all the probabilities of the actions have equal values. Also the corresponding regularization buffer is initialized.

Sensor modules

The four sensors modules mentioned above are decision blocks that calculate the response (reward/penalty), based on the last chosen action of automaton. Table 1 describes the output of decision blocks for side sensors.

As seen in Table 1, a penalty response is received from the left sensor module when the action is LEFT and there is a vehicle in the left or the vehicle is already traveling on the leftmost lane. There is a similar situation for the right sensor module.

Table -1 Outputs from the Left/Right Sensor Module

	Left/Right Sensor Module	
Actions	Vehicle in sensor range or no adjacent lane	No vehicle in sensor range and adjacent lane exists
LINE_OK	0/0	0/0
LEFT	1/0	0/0
RIGHT	0/1	0/0

The Frontal Module is defined as shown in Table 2. If there is a vehicle at a close distance (< admissible distance), a penalty response is sent to the automaton for actions LINE_OK, SPEED_OK and ACC. All other actions (LEFT, RIGHT, DEC) are encouraged, because they may serve to avoid a collision.

Table -2 Outputs from the Frontal Module

	Frontal Sensor Module	
Actions	Vehicle in range (at a close frontal distance)	No vehicle in range
LINE_OK	1	0
LEFT	0	0
RIGHT	0	0
SPEED_OK	1	0
ACC	1	0
DEC	0*	0

The reward response indicated by 0* (from the Frontal Sensor Module) is different than the normal reward response, indicated by 0: this reward response has a higher priority and must override a possible penalty from other modules
The Speed Module compares the actual speed with the desired speed, and based on the action choosed, send a feedback to the longitudinal automaton.

Table -3 Outputs from the Speed Module

	Speed Sensor Module		
Actions	Speed: too slow	Acceptable speed	Speed: too fast
SPEED_OK	1	0	1
ACC	0	0	1
DEC	1	0	0

Algorithm for the learning process
The algorithm used in implementation of the learning process is:
Step 1. Choose an action, $\alpha(n) = \alpha_i$ based on the action probability array $p(n)$.
Step 2. Compute the combined environment responses *f*.
Step 3. Update action probabilities $p(n)$ according to the defined reinforcement scheme.
Step 4. Go to step 1.
The implementation of this combination for each automaton (longitudinal respectively lateral) is showed in Table 4 (the value 0* was substituted by 2).

Table -4 Computing the response from physical environment

Longitudinal Automaton	**Lateral Automaton**

`public double reward(int action){`
`int combine;`
`combine =`
`Math.max(speedModule(action),`
`frontModule(action));`
`if (combine == 2) combine = 0;`
`return combine;`
`}`
	`public double reward(int action){`
	`int combine;`
	`combine =`
	`Math.max(leftRightModule(action),`
	`frontModule(action));`
	`return combine;`
	`}`

The values of the functions *left Right Modul e(action)*, *front Module (action)* and *speed Module (action)* are taken from the Table 1, Table 2 and respectively Table 3, where the argument represents the first column of the corresponding table.

Implementation of a simulator in a multi-agent approach using JADE
In this section is described an implementation of a simulator for the Intelligent Vehicle Control System, in a multi-agent approach. The entire system was implemented in Java, and is based on JADE platform.
JADE is a middleware that facilitates the development of multi-agent systems according to FIPA standards, and consists of [8]:

- A runtime environment in which JADE agents are living;
- A class library used by programmers (directly or through class inheritance) to build new agents;
- A collection of graphical tools to manage and monitor the activity of active agents.

The *Agent* class is the base class for user-defined agents. For this reason, from a programmer's point of view, a JADE agent is a simple instance of a Java class that extends the *jade.core.Agent* base class and overrides the *setup()* method, in which agent initialization is performed.
The computational tasks of an agent are met by "behaviors" that are executed concurrently. A behavior is a task the agent can accomplish and is implemented as an object that extends *jade.core.behaviours.Behavior*. A scheduler, internal to the *Agent* class and hidden from the programmer, manages and schedules the agent's behaviors automatically.
In order to make an agent to execute the task implemented by a behaviour object, it is sufficient to add the behaviour to the agent using the *addBehaviour()* method of the *Agent* class.
In our implementation, each vehicle has associated a JADE agent, responsible for the intelligent control of that vehicle. The coordinator agent is implemented through the *DriverAgent* class which has a behaviour called *Driving* that provide the required functionality. “Driving” means a continuous learning process, sustained by the two stochastic learning automata, namely the longitudinal automaton and respectively the lateral automaton.
Implementation of the learning process for the longitudinal automaton is described in the Figure 1:

```
public void learning(){
inti, j;
double f, h;
booleandoIt;
        // choose an action
i = getAction();
auto.showLongitudinalChoosedAction(i);
        // compute environment response
        f = reward(i);

for (int k = 1; k < HISTORY; k++)
regulation_layer[k-1]=regulation_layer[k];
if (f==0)
regulation_layer[HISTORY-1] = i;
else
          // ignore action!
regulation_layer[HISTORY-1] = -1;
doIt=true;
for (int k = 0; k < HISTORY; k++)
if (regulation_layer[k]!=i) {
doIt=false;
break;
            }
if (doIt) {
 // all action probabilities becomeequals
init();
switch(i) {
case ACC:
auto.setCurrentSpeed(auto.getCurrentSpeed()+DELTA);
auto.showCurrentSpeed();
auto.showLongitudinalExecutedAction(ACC);
break;
case DEC:
                   // do not stop!
if (auto.getCurrentSpeed()>10)
auto.setCurrentSpeed(auto.getCurrentSpeed()-DELTA);
auto.showCurrentSpeed();
auto.showLongitudinalExecutedAction(DEC);
break;
case SPEED_OK:
auto.showCurrentSpeed();
auto.showLongitudinalExecutedAction(SPEED_OK);
break;
            }
return;
        }
        h = H(i);
        // update action probabilities
        // according to the our reinforcement scheme
p[i]=p[i]+f*(-d*h)*(1.0-p[i])-(1.0-f)*(-t)*(1.0-p[i]);
for (j=0; j < ACTIONS; j++)
if (j != i)
p[j]=p[j]-f*(-d*h)*p[j]+(1.0-f)*(-t)*p[j];
}
                                          }
```

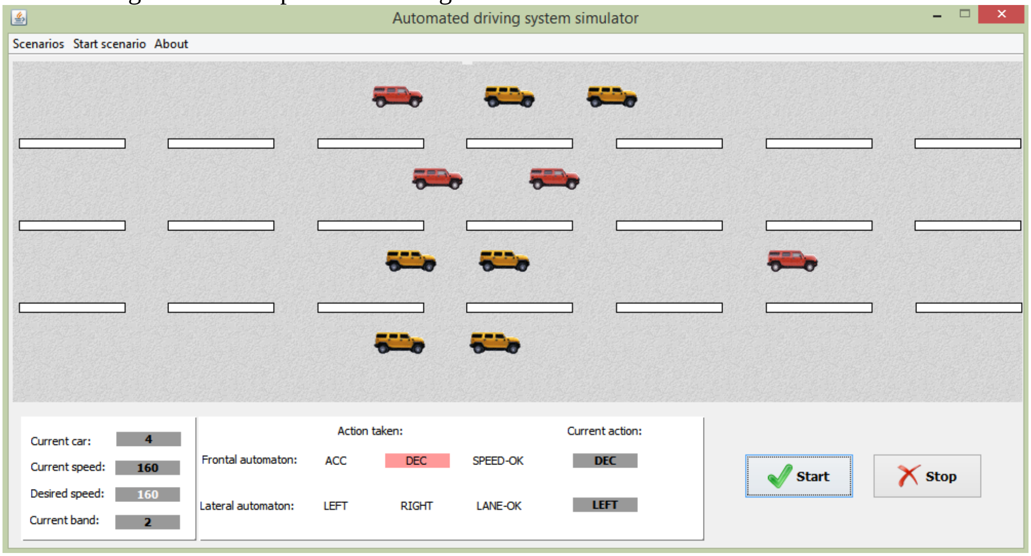
Fig. 1 The learning process for the longitudinal automaton

Function *H* from the new reinforcement scheme presented in the paper is implemented as follows:

```
// function H fromour
// reinforcement scheme
public double H(int i){
double h, temp;
  h = p[i]/(d*(1-p[i]))-eps;
  for (int j=0; j < ACTIONS; j++)
if (j!=i) {
temp=(1-p[j])/(d*p[j])-eps;
    h = Math.min(h,temp);
    }
    h = Math.max(h,0);
    h = Math.min(h,1);
return h;
}
```

Fig. 2 Function *H* from our reinforcement scheme

A snapshot of the running simulator is presented in Figure 3.


Automated driving system simulator
Scenarios Start scenario About
Start
Stop

Current car: 4
Current speed: 160
Desired speed: 160
Current band: 2

Action taken: / Current action:
Frontal automaton: ACC DEC SPEED-OK — DEC
Lateral automaton: LEFT RIGHT LANE-OK — LEFT

Fig. 3 A scenario executed in the automated driving simulator

Using the simulator interface, a user can execute the following tasks:

- Select a scenario (three scenarios are available);
- Start/stop the active scenario;
- By clicking on one of the cars, the driving parameters can be viewed and also the mode of operation of the two stochastic automata can be seen (current action, the selected action taking into account the probability of selection).

CONCLUSION

The reinforcement scheme presented in this paper satisfies all necessary and sufficient conditions for absolute expediency in a stationary environment.

Using this new reinforcement scheme was developed a simulator for an Intelligent Vehicle Control System, in a multi-agent approach. The reinforcement scheme proved its efficiency allowing the simulator to achieve its objectives: conformance with traffic parameters imposed by the user, and improving safety by minimizing crash risk.

The entire system was implemented in Java, and is based on JADE platform. The binary and source code of the simulator can be downloaded from the addresshttp://use-it.ro/simulator/.

Acknowledgements
Project financed from "Lucian Blaga" University of Sibiu research grants LBUS-IRG-2015-01.

REFERENCES

[1]. C *Rivero*, Characterization of theabsolutely expedient learningalgorithms for stochastic automata in a non-discrete space of actions, *ESANN'2003 proceedings - European Symposium on Artificial Neural Networks*, ISBN 2-930307-03-X, Bruges,Belgium, 2003, 307-312
[2]. S Lakshmivarahan, MAL Thathachar, Absolutely Expedient Learning Algorithms for Stochastic Automata, *IEEE Transactions on Systems, Man and Cybernetics*, 1973, vol. SMC-6, 281-286
[3]. C Ünsal, P Kachroo, JS Bay, Multiple Stochastic Learning Automata for Vehicle Path Control in an Automated Highway System, *IEEE Transactions on Systems, Man, and Cybernetics -part A: systems and humans*, 1999, 29 (1)
[4]. N Baba,*New Topics in Learning Automata: Theory and Applications*, Lecture Notes in Control and Information Sciences, Berlin, Germany: Springer-Verlag, 1984
[5]. C Ünsal, P Kachroo, JS Bay, Simulation Study of Multiple Intelligent Vehicle Control using Stochastic Learning Automata, *TRANSACTIONS, the Quarterly Archival Journal of the Society for Computer Simulation International*, 1997, 14 (4)
[6]. M Dorigo, Introductiontothe Special Issue on LearningAutonomousRobots, *IEEE Trans. on Systems, Man andCybernetics - part B*, 1996, 26 (3), 361-364
[7]. F Bellifemine, G Caire, T Trucco, G Rimassa, JADE Programmer's Guide, Tutorials & Guides, Web. http://jade.tilab.com/doc/programmersguide.pdf, 2010