Robot Localization Implemented with Enzymatic Numerical P Systems

Ana Brânduşa Pavel, Cristian Ioan Vasile, and Ioan Dumitrache

Department of Automatic Control and Systems Engineering,
Politehnica University of Bucharest,
Splaiul Independenţei, Nr. 313, sector 6, 060042, Bucharest, Romania
{apavel,cvasile,idumitrache}@ics.pub.ro

Abstract. Membrane computing is an interdisciplinary research field focused on new computational models, also known as P systems, inspired by the compartmental model of the cell and the membrane transport mechanisms. Numerical P systems are a type of P systems introduced by Gh. Păun in 2006 for possible applications in economics. Recently, an extension of numerical P systems, enzymatic numerical P systems, has been defined in the context of robot control. This paper presents a new approach to modeling and implementing autonomous mobile robot behaviors and proposes a new odometry module implemented with enzymatic numerical P systems for robot localization. The advantages of modeling robot behaviors with enzymatic membrane controllers and the experimental results obtained on real and simulated robots are also discussed.

Keywords: robot, membrane controller, odometry, localization, enzymatic numerical P systems.

1 Introduction

Membrane computing is a new computational paradigm inspired by the transport mechanisms of the cell's membranes. The cell is delimited by a bio-membrane and contains other compartments with specific functions (nucleus, mitochondria, etc.) [5]. The tree-like structure of the cell's membranes, which bound the compartments, and the trans-membrane transport mechanisms are the fundamental features of the membrane systems. The computational model, introduced by Gh. Păun, operates on symbols (symbolical P systems)[9,11] or variables (numerical P systems) [10]. Each membrane of the P system contains either a list of symbols or variables and a set of rules used for computation and inter-membrane communication. P systems are said to be naturally parallel and distributed systems because of their structure and their computation mechanism.

Most of the research effort has been focused on symbolical P systems. Numerical P systems (NPS) have been introduced in 2006 for possible applications in economics [10]. An extension of the NPS model, *enzymatic numerical P systems* (ENPS), has been proposed by the authors [7] in order to enhance the modeling

T.J. Prescott et al. (Eds.): Living Machines 2012, LNAI 7375, pp. 204–215, 2012.

power of NPS. In this paper, the ENPS model is used to implement an obstacle avoidance and odometric localization modules for autonomous mobile robots. The membrane system model for obstacle avoidance was proposed in [8] as an application for ENPS. Both modules have been tested on real and simulated robots and the experimental results are presented.

This paper presents a new way of modeling robot behavior using a biomimetic computational paradigm inspired by the cell's structure. To the authors' knowledge, this approach is new to the robotics field.

2 Formal Definition of the ENPS Model

2.1 NPS Model

A membrane system has a tree-like structure (figure 1) and computation takes place in parallel in all its nodes (membranes). Each membrane of a numerical P system has variables, which store information, and a set of rules (programs), which are responsible for the computation and the transfer of information between the nodes. The rules' action is inspired from chemical reactions which take place within the cell. A rule consumes the variables that are involved in it and produces a quantity that is distributed to other variables. The variables which receive new values from the rule must be contained within the current, the parent or a child membrane. This process is inspired by the membrane transport mechanisms of the cell [1] and it is responsible for the communication between the artificial membranes.

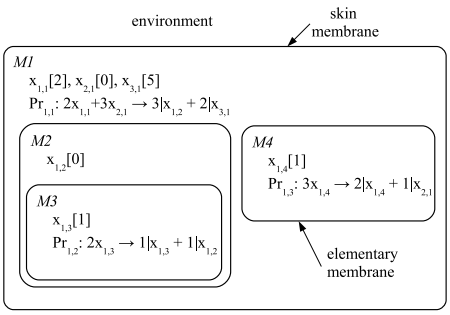


Fig. 1. Numerical P system with 4 membranes, M_1, M_2, M_3, M_4. Membrane M_1 is the skin membrane of the system and M_3 and M_4 are elementary membranes because they don't have children. Each membrane has a number of variables labeled with x_{ij} and one or no rule (program Pr_{ij}). The initial values of the variables are represented between brackets.

The ENPS model is based on the NPS model which is formally defined as follows:

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0))) \qquad (1)$$

where:

- m is the number of membranes used in the system, degree of Π; $m \geq 1$;
- H is an alphabet that contains m symbols (the labels of the membranes);
- μ is a membrane structure;
- Var_i is the set of variables from compartment i, and the initial values for these variables are $Var_i(0)$;
- Pr_i is the set of programs (rules) from compartment i. Programs process variables and have two components, a production function and a repartition protocol.
  The j-th program has the following form:

$$Pr_{j,i} = (F_{j,i}(x_{1,i}, ..., x_{k_i,i}), c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i})$$

  where:
  - $F_{j,i}(x_{1,i}, ..., x_{k_i,i})$ is the production function;
  - k_i represents the number of variables in membrane i;
  - $c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i}$ is the repartition protocol;
  - n_i represents the number of variables contained in membrane i, plus the the number of variables contained in the parent membrane of i, plus the number of variables contained in the children membranes of i.

The coefficients $c_{j,1}, \ldots, c_{j,n_i}$ are natural numbers (they may be also 0, case in which it is omitted to write "$+0|x$") [10] which specify the proportion of the current production distributed to each variable $v_1, ..., v_{n_i}$. Let us consider the sum of these coefficients: $C_{j,i} = \sum_{n=1}^{n_i} c_{j,n}$. A program $Pr_{j,i}$ is executed as follows. At any time t, the function $F_{j,i}(x_{1,i}, ..., x_{k_i,i})$ is computed. The value $q = \frac{F_{j,i}(x_{1,i},...,x_{k_i,i})}{C_{j,i}}$ represents the "unitary portion" to be distributed to variables $v_1, \ldots, v_{n_i}$, according to coefficients $c_{j,1}, \ldots, c_{j,n_i}$ in order to obtain the values of these variables at time $t + 1$. Specifically, variable v_s which belongs to the repartition protocol of program j, will receive: $q * c_{j,i}, 1 \leq s \leq n_i$.

If a variable belongs to membrane i, it can appear in the repartition protocol of the parent membrane of i and also in the repartition protocol of the child membranes of i. After applying all the rules, if a variable receives such "contributions" from several neighboring compartments, then they are added in order to produce the next value of the variable. A production function which belongs to membrane i may depend only on some of the variables from membrane i. Those variables which appear in the production function become 0 after the execution of the program.

Deterministic NPS have only one rule per membrane ($card(Pr_i) = 1$) or must have a selection mechanism that can decide which rule to apply. The NPS model with multiple rules per membrane is a non-deterministic system. However, by their structure, NPS are well suited for applications which involve numerical variables and require a deterministic behavior, such as control systems for mobile robots. Thus a selection mechanism for the active rules is defined in the ENPS model [7].

2.2 ENPS Model

ENPS is defined as a NPS with special enzyme-like variables which control the execution of the rules. Thus, an ENPS is defined as follows:

$$\Pi = (m, H, \mu, (Var_1, E_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, E_m, Var_m(0))) \quad (2)$$

where:

- E_i is a set of enzyme variables from compartment i, $E_i \subset Var_i$
- Pr_i is the set of programs from compartment i. Programs have one of the two following forms:
  1. non-enzymatic form, which is exactly like the one from the standard NPS: $Pr_{j,i} = (F_{j,i}(x_{1,i}, \ldots, x_{k_i,i}), c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i})$
  2. enzymatic form: $Pr_{j,i} = (F_{j,i}(x_{1,i}, \ldots, x_{k_i,i}), e_{t,i}, c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i})$, where $e_{t,i} \in E_i$

The enzymatic mechanisms of the ENPS model is used for the selection of the valid rules. The enzyme-like variables are inspired by biological enzymes, which are molecules that control most of the biochemical process in living cells. By catalyzing reactions, enzymes synchronize the steps of a biological process.

To understand how ENPS work, let us consider one membrane $M1$ with the following variables: $x_{11}[3]$, $x_{21}[2]$, $e_{11}[4]$ and one production function: $2 * x_{11} + x_{21}(e_{11} \rightarrow)$, where one may notice a specific variable attached, e_{11}, which is the enzyme (figure 2).

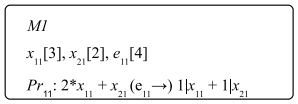


Fig. 2. Membrane with an enzyme variable

In this case the condition is $e_{11} > min(x_{11}/2, x_{21})$, but because $min(x_{11}, x_{21}) > min(x_{11}/2, x_{21})$, the simplified and more general condition $e_{11} > min(x_{11}, x_{21})$ ensures that the amount of enzyme is more than enough and that the reaction can take place. Thus, a rule is active if the associated enzyme variable has a greater value than the minimum of the variables involved in the production function. Because the values of the variables can be sometimes negative, in some applications it is more convenient to test if the value of the enzyme is greater than the absolute value of one of the variables contained in the production function. For example, the rule $Pr_{1,1}$ in membrane $M1$ (figure 2) is active if $e_{11} > min(|x_{11}|, |x_{21}|)$. This last condition is used in this paper. There can be more than one active rule in a membrane or none. In a computational step, all active rules in all membranes are executed in parallel. The universality of the NPS and ENPS computational models is proven in [10] and [14].

3 ENPS Controllers

Membrane controllers can be used to control autonomous mobile robots and to generate various desired behaviors and cognitive abilities, like obstacle avoidance, localization, moving to a given position, wall following, following another robot, etc. The major advantage of using NPS as a modeling tool is that P systems are naturally parallel and distributed systems. Membranes of a NPS can be distributed over a grid or over a network of microcontrollers in a robot. The computation done in each membrane region (the execution of a membranes rules) can also be done in parallel [3].

Another important property is that membranes can only communicate with their parents and their children membranes. Thus, communication in distributed P systems can be implemented efficiently.

Furthermore, membrane controllers can be integrated easily in the control program of the robotic system. Membrane systems can be added or changed to embed the desired functionality without changing the code of the control program. The membrane system definition can be stored in separate files (for example in xml format) and loaded as needed. A simulator for ENPS systems is necessary to execute the membrane systems. The simulator can, however, be optimized for the platform it runs on, taking advantage of the underlying hardware, microcontroller/procesor/DPS, memory architecture and communication technology (TCP/IP, I2C, TWI, Bluetooth, etc.). For instance, a parallelized GPU-based simulator for ENPS was recently proposed in [4].

A robot controller which uses two modules, obstacle avoidance and localization, is presented in Pseudocode 3. In each loop of the controller, the sensors are read (infrared and motor encoders), then the position of the robot is updated based on the previous position and on the values of the encoders. The proximity sensors are used to compute the motors' speeds such that obstacles are avoided. Finally, the motors' speeds are sent to the robot.

```
while(True) {
 read_sensors()
 position = odometry(position, encoders)
 motors_speeds = avoid(proximity_sensors)
 set(motors_speeds)
}
```

This paper proposes a new ENPS module for odometric localization which was implemented and tested on simulated e-puck and KheperaIII robots and on real e-puck robots. Experimental results obtained so far for an autonomous robot with both obstacle avoidance behavior and localization ability will further be presented. The ENPS model for obstacle avoidance is detailed in [8].

4 Odometric Localization for an Autonomous Mobile Robot

An autonomous mobile robot should be able to know its position at any time. Localization is one of the most important and difficult problems in autonomous mobile robotics. Different localization systems used to determine the position of an autonomous robot at any time are presented in [12,13]. Localization can be implemented in many ways, using different devices like: the encoders of the motors, accelerometers, beacons, GPS [13].

An ENPS model which implements odometric localization for an autonomous mobile robot, using the information received from the motors' encoders, is proposed. The ENPS for odometric localization has been designed as a membrane system with 5 membranes (figure 3). The enzyme-like variables have an essential role in the control of the program flow, synchronization and parallel computation. As the robot must know its position at any time, the module receives in the begining of each cycle of the controller the following input information: the initial position of the robot (x_i, y_i, θ_i), where x_i, y_i, represent the coordinates in reference coordinate system and θ_i is the angle made by the direction of the robot and the x axis counterclockwise (the orientation of the robot). Also, the distances traveled by each wheel are input values: dL for the left wheel and dR for right wheel. The output of the module is the updated position of the robot: (x_f, y_f, θ_f) in the same coordinate system. The trajectory of the robot on a short distance can be approximated with a circular arc. Each wheel travels a distance which is given by its encoder. The encoder returns the number of steps which can be converted into a distance. Thus, the following position update formulas are obtained for a differential wheeled robot [13]:

$$\Delta\theta = \frac{d_R - d_L}{wheelDist}$$
$$\Delta s = \frac{d_R + d_L}{2}$$
$$\Delta x = \Delta s \cdot \cos(\theta + \frac{\Delta\theta}{2})$$
$$\Delta y = \Delta s \cdot \sin(\theta + \frac{\Delta\theta}{2})$$

The updated position of the robot, (x_f, y_f, θ_f), is computed in the following way:

$$x_f = x_i + \Delta x \qquad y_f = y_i + \Delta y \qquad \theta_f = \theta_i + \Delta\theta$$

The position of the robot is measured in the point situated in the middle of the segment which joins the two wheels. The distance between the two wheels of the robot is a fixed parameter that depends on the type of robot and is passed to the membrane system via the b variable in membrane *Odometry* (figure 3).

The membranes *Cosine* and *Sine* approximate the values of sine and cosine using their power series:

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!} \qquad \cos(x) = \sum_{n=0}^{\infty} (-1)^n \cdot \frac{x^{2n}}{(2n)!}$$

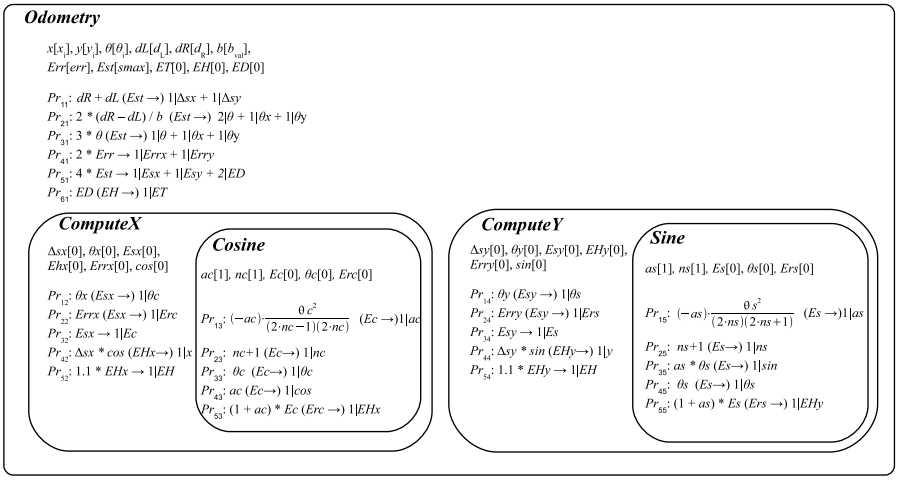


Fig. 3. Enzymatic membrane system for odometric localization

The general term of the power series is computed recursively, in rule $Pr_{1,3}$ for *Cosine* and in rule $Pr_{1,5}$ for *Sine*, and added to the final value, variable *cos* from membrane *ComputeX* and *sin* from *ComputeY*. After a number of steps, both *Cosine* and *Sine* finish their computations. The two membranes are executed in a number of steps which depends on the input value, θc for *Cosine* and θs for *Sine*, and the error, Erc, respectively Ers. When the general term of the series becomes less than the given error parameter, the computation stops.

In *ComputeX*, the value of Δx is computed by the production function $Pr_{4,2}$ which is activated by Ehx. The final value of x is computed by summing Δx, received from membrane *ComputeX*, with the initial value, x_i. Similarly, the value of y is computed. Then the values of EHx and EHy are transferred to EH ($EH \leftarrow 1.1 \cdot EHx$; $EH \leftarrow 1.1 \cdot EHy$). The factor 1.1 has been chosen in such a way that, when both membranes *ComputeX* and *ComputeY* finish their computations, the enzyme EH from the membrane *Odometry* have a greater value than the variable ED from the same membrane. EH enzyme from the skin membrane sums the values of EHx from *ComputeX* and EHy from *ComputeY*. If both membranes finished their computations, EH has a value close to $2.2 \cdot smax$, while ED is $2 \cdot smax$. It is important to note that in general membranes *ComputeX* and *ComputeY* do not finish at the same time. When EH becomes greater than ED, the rule $Pr_{6,1}$ from the membrane *Odometry* is executed, therefore varible ET receives a positive value which generates the termination of the program which returns the current position of the robot, stored in variables x, y and θ.

The two membranes, *ComputeX* and *ComputeY*, are synchronized in the skin membrane, *Odometry*, by the enzymatic mechanism. If only one of the two membranes finished the computation, then $ED = 2 \cdot smax > EH = 1.1 \cdot smax$, so the rule $Pr_{6,1}$ is not active yet. It is activated only when both membranes have finished their computations and $ED = 2 \cdot smax < EH = 2.2 \cdot smax$.

The odometric localization implemented with ENPS is a parallel computation structure. The enzymatic mechanism enhances the computation power of the membrane system and also allows the control of the program flow and synchronization between the parallel computations.

5 Advantages of ENPS Model

Both NPS and ENPS models can be used for modeling autonomous mobile robot behaviors. The numerical nature, the distributed and parallel structure and the computing power, make membrane controllers suitable candidates for robotic control systems.

ENPS controllers have a less complex structure than NPS controllers. Designing a classical NPS controller is difficult and requires a lot of tricky design mechanisms. If the order of the generated values is changed, the system would not work. By using enzyme-like variables, the model of the controller is clearly simplified, easier to implement and more efficient than the one modeled with classical NPS. Enzyme variables control the program flow. Therefore, they can be used for conditional trans-membrane transport, as stop conditions and synchronization mechanism. In ENPS, if the result is generated, the computational process stops due to the stop conditions implemented by the enzymatic mechanism, while in NPS all the membranes have to finish all their computations in a given number of steps. The design of an ENPS controller requires less effort and the performance of the controller is increased by reducing the computational procedure. Enzyme variables can also filter the noise from the sensors. In the ENPS controller for obstacle avoidance [8], the values lower (greater, after rescaling) than a given number are ignored because they are not considered to indicate a real obstacle detection.

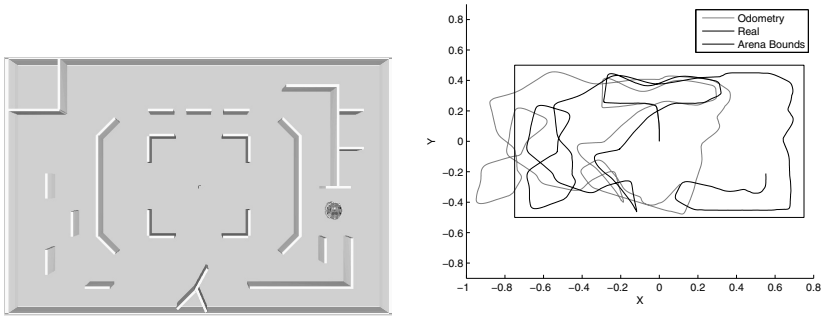
The membrane representation is an advantage for both NPS and ENPS because membrane structures are very efficient for designing and modeling robotic behaviors in a parallel and distributed manner. The controllers are designed independent of how the membrane system is distributed and executed in parallel. Only the simulator for ENPS structures is responsible for the parallel and distributed execution of the membranes. The enzymatic mechanism provides stop conditions, rules selection conditions, and synchronization between the computations performed in different membranes.

6 Experiments and Results

A Java simulator, SimP, which computes ENPS and NPS structures has been implemented in order to test the membrane controller models on robots [6]. Other simulators which can be used to execute ENPS models are SNUPS, which is free and has a graphical user interface [2], and the parallelized GPU-based ENPS simulator proposed in [4]. ENPS is an extension of NPS, thus the simulators can execute NPS structures as well. The membrane controllers are stored in xml files which are parsed by SimP simulator. Different behaviors can be stored in xml

format which is a uniform representation. XML representation does not depend on the implementation of the simulator or of the robotic system which integrates the controllers. Thus, implementation changes in the membrane simulator or in the robotic system do not interfere with the membranes. If the membrane simulator is optimized, the performance of all membrane controllers increases.

A framework has been developed in order to test the ENPS modules on real and simulated robots. The framework transfers the information received from the sensors (infrared, motors' encoders) to the membrane controllers. The membrane structures are then simulated with SimP and return the output information (motors' speeds, robot position and orientation) to Webots robotics simulator. Using Webots to route the control program on real robots, the behaviors have been tested on real robots as well.



(a) Arena

(b) Odometry - real trajectory and computed trajectory

Fig. 4. Execution time of the controllers' cycle in simulated experiments

The ENPS modules have been successfully tested on both simulated KheperaIII and e-puck robots and on real e-puck robots. Multiple tests have been performed with one robot and more robots in an arena with obstacles. An example of a simulated experiment is shown in figure 4(a). In all experiments, the robots avoided all the obstacles in the arena, updating their position.

In figure 5, the maximum value read by the sensors ($S_{max} = max(S_1, \ldots, S_8)$) and the speeds of the two wheels are displayed for each cycle (a cycle corresponds to one loop of the controller presented in Pseudocode 3). The peaks in the graph of maximum sensors' value (top plot of figure 5) represent detected objects and generate the corresponding spikes in the two graphs of the motors' speeds because the robot has to turn in order to avoid the obstacle. Other interesting features in the maximum sensors' graph are regions of non-zero, near constant values which correspond to the case when the robot passes through a narrow corridor. One such example can be noticed around the 10000 cycle. In this case, the speeds of the motors do not present spikes and are close to the value of the cruise speed which is 200. Therefore, the controller is able to stabilize the trajectory of the robot while passing through tunnels. The mean absolute values of the two speeds are very close to the cruising speed as shown table 1.

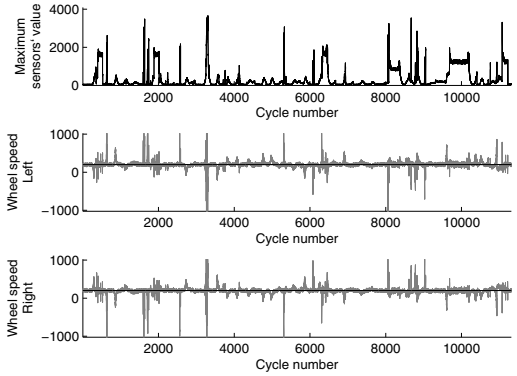


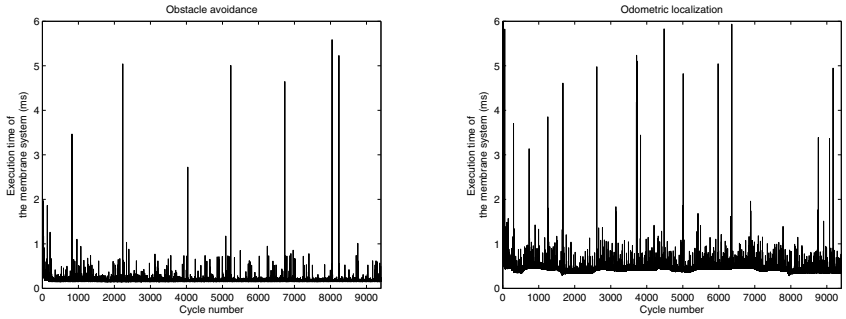
Fig. 5. Maximum sensors' value and motors' speeds in each cycle

Figure 4(b) illustrates the real trajectory of the robot obtained from Webots simulator and the one computed using the odometry module in an experiment done in the arena shown in figure 4(a). In order to prove that the odometry localization module implemented with ENPS returns good estimates of the position, the values computed by the membrane module have been compared to the values returned by the Webots simulator. The positioning error is low at the beginning of the experiment, but increases in time due to numerical and approximation errors and wheel slippage. As shown in figure 4(b), the two trajectories (real and computed) of the robot are similar, but the computed one is shifted to the left. The robot has traveled a total time of 362 seconds and a distance of 8.9849 meters as shown in table 1. The positioning error at the end of the experiment is 0.4943 meters.

The execution time of both modules (avoid and odometry) is very low compared to the cycle time of the robot controller. The cycle time is defined as the duration of a loop (Pseudocode 3) which is about 32 ms in Webots simulator for KheperaIII and e-puck robots and about 180 ms for real e-puck robots. The cycle time in the real experiments is greater because it includes the bluetooth communication. In figure 6 the execution time of both membrane modules for each robot controller cycle is represented. For obstacle avoidance the mean execution time is 0.1884 ms and for the odometric localization, the mean execution time is 0.4719 ms (table 1). The designed membrane controllers can be used with other differential wheeled robots as well, by changing the weights of the sensors for obstacle avoidance and the distance between the wheels for the odometric localization module. The weights used in the avoid procedure depend on the type of sensors and on their placement around the robot. Thus, membrane controllers can be successfully used in autonomous mobile robot applications.

Table 1. Summary of results from experiments with simulated

Avoid (9 membranes)	Execution time	mean	0.1884 ms
		stddev	0.1441 ms
		max	5.5870 ms
		min	0.1460 ms
	S_{max}	mean	316.7189
		stddev	485.7635
		max	3647.0
	Speed left	mean	213.9461
		stddev	144.3022
	Speed right	mean	206.8451
		stddev	144.3022
Odometry (5 membranes)	Execution time	mean	0.4719 ms
		stddev	0.2351 ms
		max	7.7400 ms
		min	0.2830 ms
Total distance traveled		8.9849 m	
Total time traveled		362 s	
Final positioning error		0.4943 m	



(a) Obstacle avoidance (b) Odometric localization

Fig. 6. Execution time of the controllers' cycle in simulated experiments

7 Conclusions and Future Work

Two ENPS modules for obstacle avoidance and odometric localization have been designed and tested on real and simulated robots. By using a suitable simulator for ENPS, the membrane systems are distributed over computational nodes (like microcontrollers in a robot) in a transparent way. The programmer does not have to worry about communication and synchronization issues or how to distribute the system.

Future work include further testing of the proposed ENPS controllers integrated in the local agent of a cognitive multi-robot architecture. Other robot behaviors will also be modeled with ENPS and tested on autonomous robots. It is a research goal of the authors to develop different components of a robotic system: localization, navigation, planning, etc as membrane systems which will run on a virtual machine distributed over networked microcontrollers. The virtual machine will run the membranes (the code) in a distributed and parallel way, transparently from the user in a similar way as Java and Python. It is a current effort to also develop standard libraries that provide basic and often used functionalities, such as a math library (for sine, cosine, etc.). In this regard, the

long term objective is to create a controller programming environment that is completely distributed and parallel.

References

1. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: Molecular Biology of the Cell, 4th edn. Garland Science, NY (2002)
2. Arsene, O., Buiu, C., Popescu, N.: Snups – a simulator for numerical membrane computing. Intern. J. of Innovative Computing, Information and Control 7(6), 3509–3522 (2011)
3. Buiu, C., Vasile, C.I., Arsene, O.: Development of membrane controllers for mobile robots. Information Sciences 187, 33–51 (2012)
4. Garcìa-Quismondo, M., Pèrez-Jimènez, M.J.: Implementing enps by means of gpus for ai applictions. In: Proceedings of Beyond AI: Interdisciplinary Aspects of Artificial Intelligence (BAI 2011), Pilsen, Czech Republic, pp. 27–33 (December 2011)
5. Lodish, H., Berk, A., Kaiser, C.A., Krieger, M., Scott, M.P., Bretscher, A., Ploegh, H., Matsudaira, P.: Molecular Cell Biology (Lodish, Molecular Cell Biology), 6th edn. W.H. Freeman (June 2007)
6. Pavel, A.B.: Membrane controllers for cognitive robots. Master thesis, Department of Automatic Control and Systems Engineering, Politehnica University of Bucharest (February 2011)
7. Pavel, A.B., Arsene, O., Buiu, C.: Enzymatic Numerical P Systems - A New Class of Membrane Computing Systems. In: The IEEE 5th International Conference on Bio-Inspired Computing: Theory and Applications, Liverpool, UK (2010)
8. Pavel, A.B., Buiu, C.: Using enzymatic numerical P systems for modeling mobile robot controllers. Natural Computing (in press), doi: 101007/s11047-011-9286-5
9. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
10. Păun, G., Păun, R.: Membrane computing and economics: Numerical p systems. Fundamenta Informaticae 73, 213–227 (2006), http://portal.acm.org/citation.cfm?id=1231159.1231179
11. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York (2010)
12. Siciliano, B., Khatib, O. (eds.): Springer Handbook of Robotics. Springer (2008)
13. Siegwart, R., Nourbakhsh, I.R.: Introduction to Autonomous Mobile Robots. Bradford Company, Scituate (2004)
14. Vasile, C.I., Pavel, A.B., Dumitrache, I., Păun, G.: On the Power of Enzymatic Numerical P Systems (submitted)