Development of membrane controllers for mobile robots

Catalin Buiu[a,b,*], Cristian Vasile[a], Octavian Arsene[a]

[a]*Laboratory of Natural Computing and Robotics, Politehnica University of Bucharest, Spl. Independentei 313, 060042 Bucharest, Romania*
[b]*Faculty of Biology, University of Bucharest, Spl. Independentei 91-95, Bucharest, Romania*

Abstract

The main contribution of this paper is the introduction of the new concept of membrane controller based on the structure and functioning of a deterministic numerical P system. The procedure for developing a membrane controller and for using it to control a mobile robot is explained and several test cases are given in which membrane controllers are used to control both simulated and real mobile robots and to generate various desired behaviours (obstacle avoidance, wall following, and follow the leader). The experiments reported in this paper validate the concept and prove that the performance of a membrane controller is comparable to or better than that of other controllers (such as fuzzy logic controllers).

Keywords:
bio-inspired computing, membrane systems, numerical P system, membrane controller, mobile robot, simulation

1. From biomembranes to membrane computing

Living cells are admirable examples of functional complexity and robustness. Cells can grow, reproduce, respond to stimuli, process information, and carry out an amazing array of chemical reactions, and all these abilities define life [7]. A cell is a wonderful and complex biochemical machine. The

*Corresponding author
Email addresses: `cbuiu@ics.pub.ro` (Catalin Buiu), `cvasile@ics.pub.ro` (Cristian Vasile), `oarsene@ics.pub.ro` (Octavian Arsene)
URL: `http://natural.ics.pub.ro` (Catalin Buiu)

architecture of a cell is now better understood and the important role of the biomembranes as separators and filters is acknowledged.

Bioinspired computing is that branch of information sciences aiming at developing new problem solving techniques and computing models based upon the structure and functioning of living systems. Major paradigms in bioinspired or natural computing are neural networks, evolutionary computing, swarm intelligence, DNA computing, among others. Membrane systems (or P systems as they are generically known) represent an important research direction in bioinspired computing and was initiated a decade ago by Gheorghe Paun [8].

A P system (PS) is a computational model based upon the structure of an eukaryotic cell and it mimicks the interaction and evolution of chemicals inside of biomembranes [12]. The architecture (or membrane structure) of a PS is a hierarchical arrangement of membranes, in which the outermost membrane is called the skin membrane and separates the system from its environment. This skin membrane is analogous to the plasma membrane of a living cell. The membranes define regions as the biomembranes define working compartments. A membrane is called elementary if it has no other membrane inside. For a sample membrane structure, refer to Fig. 1.

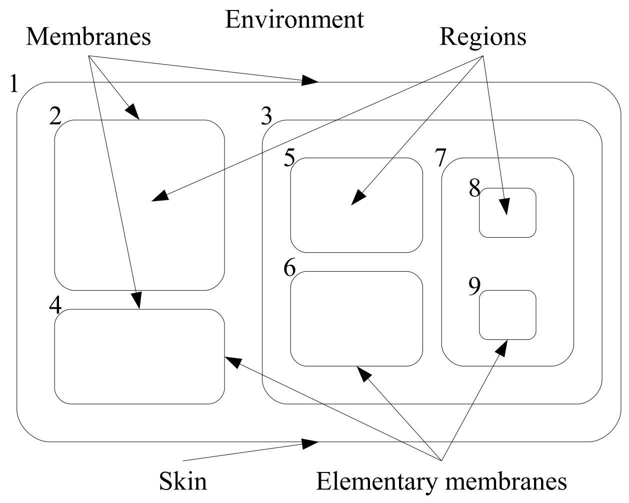


Figure 1: A typical membrane structure of a P system

In every region there is a multiset of objects (abstract counterparts of real chemicals, such as ions, proteins, etc.) and a set of evolution rules (abstract chemical reactions), see Fig. 2. A typical evolution rule for a region r of a PS is of the form $a \to b_{inj} d_{out} d_{here}$ [11]. This rule reads as follows: a copy of object a will be replaced by a copy of object b and two copies of object d. More, the rule specifies the destination of the products: the copy of b will enter the inner membrane labeled by j (b_{inj}), a copy of d will pass through the current membrane and enter the outside region (d_{out}), and a copy of d will remain in region r (d_{here}). This rule can be applied if the current region r includes a membrane (region) j. So, there are three basic operating ways for a rule: 1. *here* rule: the output objects are passed into the current membrane (region); 2. *in* rule: the outputs will be passed to one or more children membranes; 3. *out* rule: the outputs will be passed to an upper membrane in the hierarchy or to a sibling membrane.

The evolution rules are typically applied in a non-deterministic way and in a maximally parallel way: the order of rule application is chosen at random, and respectively, all possible rule assignments must take place during every step of the computation. As an example, the P system from Fig. 2 which is generating the Fibonacci numbers is iterated for a few steps. Membrane *0* is the output membrane and has no evolution rules inside. The result is represented by the number of copies of the object e in the output membrane.

Figure 2: A P system for generating Fibonacci numbers

Membrane systems are synchronous and a computation proceeds from an initial state (initial chemicals existing in one of the membranes) to an end state (when no rule can be applied anymore) through a number of discrete steps or transitions between configurations of the system. A configuration is halting if no rule is applicable in any region, and a computation is halting if it reaches a halting configuration. The result of a (halting) computation is the number of objects sent to the environment (through the skin membrane) during the computation [8].

An important number of computation mechanisms and classes of PS are discussed in the literature (membrane charge, tissue P systems, symport and antiport-based communication through membranes, catalytic objects, division of membranes, membrane algorithms, etc.). More details are given in [9], and a recent and comprehensive overview of the field is [13]. An updated bibliography, list of simulators and open problems in PSs research can all be found at the web address http://ppage.psystems.eu.

Current applications of PS cover a wide range: modeling of biological and biochemical processes, linguistics, cryptography, economics, optimization, modeling intercellular communication mechanisms, interspecies dynamics in ecological systems, etc. [13, 16]. Applications in process control are begining to emerge, see for example [5] where a membrane systems-based control strategy is used for designing stable controllers for unstable varying plants.

There are many theoretical results related to the computational power and complexity of PS [13]. For example, it has been proven that deterministic PSs can be simulated on a Turing machine in a polynomial time and variants of PSs have been shown to be able to solve NP complete problems in polynomial time [11].

Numerical PSs have been introduced in 2006 as possible models of economic and business processes and despite powerful mathematical results concerning this class of PSs, no concrete applications have been proposed. For this reason, in [13, chap. 23.8], they are considered an "exotic" subclass of PSs. As its main contribution, this paper will introduce the new concept of membrane controllers based on using numerical PSs and demonstrate the use of such controllers for controlling a mobile robot. Next section will present the fundamentals of numerical PSs, while section 3 will present the idea of using membrane systems as computational blocks of cognitive architectures. Section 4 will present the structure of the proposed membrane controllers for three desired robot behaviours. Section 5 will give the experimental results obtained both in simulation and on real robots. Performance indices will be defined and analyzed for each membrane controller. The performance of the proposed membrane controller will be compared to that of a Mamdani fuzzy controller. Section 6 will conclude the paper with discussions and directions for further improvements.

2. Numerical P systems

The focus of this paper is on numerical P systems (NPS), which will be described in the following. They have been introduced in [10] with an inspiration from economic and business processes. The architecture of an NPS is identical to the architecture of a symbolic P system described above (see Fig. 1), but in this case there are numerical values (variables) instead of symbols and each membrane usually has a program consisting of a production function and a repartition protocol, see Fig. 3.

Figure 3: A simple numerical P system

The local variables have initial values (integer or real numbers). A production function takes the local variables and computes a number. This number will be distributed among the variables within various membranes based on the repartition protocol. During each computation step every region's production function is executed in parallel with the other ones. The production function is a polynomial function that uses integer/real coefficients. In case of multiple programs in a membrane, there are two possible ways to run the system: 1. to randomly select one program and execute it or 2. to control the execution by using special variables which play the role of biological enzymes (this enzymatic variant keeps the system deterministic and is described with much details in [14]).

The repartition protocol takes the form: $c_1|v_1 + c_2|v_2 + ...c_i|v_i + ... + c_n|v_n$, in which c_i are natural numbers and specify the proportion of the current production distributed to each variable v_i. More concrete, there is a unitary portion, q, distributed to variables v_i and which is calculated as the value of production function value at the current time divided by the sum of c_i coefficients from each repartition protocol. Then, a variable v_i from the distribution list will receive the value $q * c_i$. In case of a variable which gets more than one contribution from several membranes, they are added in order to produce the next value.

A variable x_{ij} is 'productive' if it does appear in a production function, then is 'consumed' and reset to zero, otherwise the initial value is added to the received contributions. The values of the variables at next time step are computed by using repartition protocols, and so, portions distributed to variables are added to form the new value.

For exemplification, one may consider a simple NPS structure as in Fig. 3 where there are two membranes with two variables each. One may note that x_{22} is not productive as it does not appear in any of the two production functions and so it will be not reset to zero at each computation step. In square brackets are given the initial values for all four variables of interest:

- Membrane 1:
  - variables: x_{11} has an initial value of 1, x_{12} has an initial value of 2;

  - production function: $F_1 = 4 * x_{11} + x_{12}$;
  - repartition protocol: x_{11} receives 1 ($c_{11} = 1$), x_{12} receives 1 ($c_{12} = 1$), x_{22} receives 1 ($c_{13} = 1$).

- Membrane 2:
  - variables: x_{21} has an initial value of 3, x_{22} has an initial value of 1;
  - production function: $F_2 = 3 * x_{21} - 3$;
  - repartition protocol: x_{11} receives 1 ($c_{21} = 1$), x_{22} receives 1 ($c_{22} = 1$), x_{21} receives 1 ($c_{23} = 1$).

Let's suppose the entire evolution cycle of the NPS from Fig. 3 has 4 steps, of which the first two are detailed below.

- Step 1:
  - Membrane 1:
    - $x_{11} = 1, x_{12} = 2$;
    - compute production function's value, $F_1 = 4 * x_{11} + x_{12} \Rightarrow F_1 = 6$;
    - compute 'unitary portion' $q_1 = F_1/(c_{11} + c_{12} + c_{13})$ to be distributed to variables x_{11}, x_{12} and $x_{22} \Rightarrow q_1 = 2$
  - Membrane 2:
    - $x_{21} = 3, x_{22} = 1$;
    - compute production function's value, $F_2 = 3 * x_{21} - 3 \Rightarrow F_2 = 6$;
    - compute 'unitary portion' $q_2 = F_2/(c_{21} + c_{22} + c_{23})$ to be distributed to variables x_{11}, x_{21} and $x_{22} \Rightarrow q_2 = 2$;
  - Compute the new variables' values for Membrane 1 and Membrane 2 - cycle 1:
    - x_{11} appears in both repartition protocols and is productive, so: $x_{11} = c_{11} * q_1 + c_{21} * q_2 = 2 + 2 \Rightarrow x_{11} = 4$;
    - x_{12} appears in the repartition protocol from Membrane 1 and is productive, so: $x_{12} = c_{12} * q_1 \Rightarrow x_{12} = 2$;

    * x_{21} appears in the repartition protocol from Membrane 2 and is productive, so: $x_{21} = c_{23} * q_2 \Rightarrow x_{21} = 2$;
    * x_{22} appears in both repartition protocols. As a 'non-productive' variable, x_{22} does not reset to 0 after each cycle, so it is updated as: $x_{22} = 1 + c_{13} * q_1 + c_{22} * q_2 = 1 + 2 + 2 \Rightarrow x_{22} = 5$

- Step 2:
  - Membrane 1:
    * $x_{11} = 4, x_{12} = 2$;
    * compute production function's value, $F_1 = 4 * x_{11} + x_{12} \Rightarrow F_1 = 18$;
    * compute 'unitary portion' $q_1 = F_1/(c_{11} + c_{12} + c_{13})$ to be distributed to variables x_{11}, x_{22} and $x_{21} \Rightarrow q_1 = 6$;
  - Membrane 2:
    * $x_{21} = 2, x_{22} = 5$;
    * compute production function's value, $F_2 = 3 * x_{21} - 3 \Rightarrow F_2 = 3$;
    * compute 'unitary portion' $q_2 = F_2/(c_{21} + c_{22} + c_{23})$ to be distributed to variables x_{11}, x_{21} and $x_{22} \Rightarrow q_2 = 1$;
  - Compute the variables for Membrane 1 and Membrane 2 - cycle 2:
    * x_{11} appears in both repartition protocols and is productive, so: $x_{11} = c_{11} * q_1 + c_{21} * q_2 = 6 + 1 \Rightarrow x_{11} = 7$;
    * x_{12} appears in the repartition protocol from Membrane 1 and is productive, so: $x_{12} = c_{12} * q_1 = 1 * 6 \Rightarrow x_{12} = 6$;
    * x_{21} appears in the repartition protocol from Membrane 2 and is productive, so: $x_{21} = c_{23} * q_2 = 1 * 1 \Rightarrow x_{21} = 1$;
    * x_{22} appears in both repartition protocols and is not productive, so x_{22} does not reset to 0 after each cycle and: $x_{22} = 5 + c_{13} * q_1 + c_{22} * q_2 = 5 + 6 + 1 \Rightarrow x_{22} = 12$.

Using the same reasoning for steps 3 and 4, the results are:

- Step 3: $x_{11} = 11.34$; $x_{12} = 11.34$; $x_{21} = 0$; $x_{22} = 23.34$;
- Step 4: $x_{11} = 17.89$; $x_{12} = 18.89$; $x_{21} = -1$; $x_{22} = 41.43$.

Fig. 4 presents in a graphical way the evolution of the four variables during the computation steps.

Figure 4: Evolution of variables during 4 computation steps for the NPS in Fig. 3

To summarize, a numerical P system can be formally expressed as in [10]:

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), ..., (Var_m, Pr_m, Var_m(0))) \tag{1}$$

where:

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

$$Pr_{l,i} = (F_{l,i}(x_{1,i}, ..., x_{k_i,i}), c_{l,1}|v_1 + ... + c_{l,n_i}|v_{n_i}) \tag{2}$$

As for the computing power of NPSs, in [10] it is demonstrated that "non-deterministic systems of this type, using polynomial production functions, characterize the Turing computable sets of natural numbers, while deterministic systems, with polynomial production functions having non-negative coefficients, compute strictly more than semilinear sets of natural numbers".

The first simulator for NPS (SNUPS) has been developed by the authors of this paper. The steps to be followed for running a simulation in SNUPS are [2]:

1. Building the membrane structure;
2. Adding variables to a membrane;
3. Creating rules;
4. Starting computation;
5. Checking computation results;
6. Saving the membrane structure.

The simulator's functionalities and related use cases are further presented in the SNUPS user manual [15] and in [2].

3. Membrane controllers - a new application of P systems

Complex real-world systems, such as aerospace systems or autonomous robots, require a control system which should be capable of responding intelligently and autonomously to gaps in its knowledge and to contexts that have not been specified in the initial design. In designing the control architectures for complex systems, there was a long-term tendency to mimic the organization and functional complexity of the human brain. Cognitive modeling is the process of developing computational models of cognitive processes with the aim of understanding human cognition and cognitive architectures are cognitive models that are domain-generic and should exhibit a wide range of cognitive abilities, such as perception and action, memory, learning, etc. [1]. A well known research initiative concerned the development of biologically inspired cognitive architectures [3]. DARPA's Broad Area Announcement (BAA), issued in 2005, asked for theories and applications aimed at "implementing computational models of human cognition that could eventually be used to simulate human behaviour and approach human cognitive performance in a wide range of situations".

The use of PSs as computational modules of cognitive architectures was first proposed in [1]. In fact, the cognitive architecture as a whole can be considered as a membrane, while various modules of the architecture (execution, memory, planning, learning etc.) can be implemented by dedicated membranes, either symbolic, or numerical. So, the system becomes a hybrid membrane that can be globally considered as a three level architecture which can be mapped onto the gross anatomy and functionality of the human brain (Fig. 5).

The focus of this paper is on the execution level of the proposed architecture, where we introduce the new concept of membrane controller (MC).

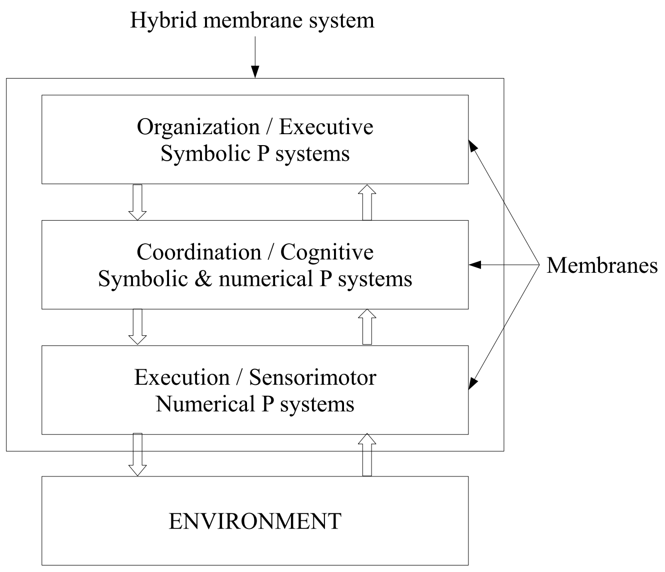


Figure 5: Cognitive architectures as hybrid membrane systems

A MC is an NPS that directly interacts with the process to be controlled, receiving sensory inputs from the environment (e.g., distance values from infrared sensors of a robot) and sending commands to the environment (e.g., desired speeds to the wheels of the robot), according to a control law that the MC is implementing such that the evolution of the process (robot) proceeds as desired. For example, a MC may assure the desired simultaneous obstacle avoidance and exploring behaviours of a robot.

Given a process F to be controlled, with inputs u, outputs y and predefined setpoints r, a membrane controller (MC) is a construct of the following form:

$$MC = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), ..., (Var_m, Pr_m, Var_m(0)), y, u) \quad (3)$$

where:

- m is the number of membranes, $m \geq 1$;
- H is an alphabet that contains m symbols (the labels of the membranes);

- μ is a membrane structure;
- Var_i is a set of variables from compartment i;
- $Var_i(0)$ is the set of initial values for the variables in region i;
- Pr_i is the program from compartment i, composed of a polynomial production function and a repartition protocol, chosen such that the process outputs y match the predefined setpoints r;
- y is a set of input variables of the MC (from a subset of regions);
- u is a set of output variables (from a subset of regions) which provide the result of the computation.

As mentioned before, this MC has to be understood as part of a hierachical cognitive architecture, and it may receive higher level commands from the upper levels which are also membranes (using the communication rules from the membrane systems theory). So, by its own structure, which is membrane-based, the MC can be easily interfaced with the other modules and integrated into the global cognitive architecture, and this is one of the main advantages of the proposed approach to use PS as building blocks for cognitive/control architectures.

Furthermore, PSs are naturally parallel and distributed systems. For example, membranes of an NPS can be distributed over a grid or over a network of microcontrollers in a robot. The computation done in each membrane region (i.e. the execution of a membrane's rules) can also be done in parallel. This is very important, because given a membrane system, which is an abstract implementation (membrane program) of a desired behaviour, it can be executed in a distributed and parallel way without having to worry about the design and implementation issues regarding parallelization and distribution. In this sense, NPSs can be used as a modeling paradigm for parallel and distributed control systems. There is also another important property of PSs which is given by the tree structure of the membranes. Because membranes can only communicate with their parents and their children membranes, communication in the distributed PS can be efficiently implemented.

While there are no physical implementations of a PS yet, the properties of NPS make them suitable for hardware implementation. A generic computational node could be implemented to simulate a single membrane. These nodes can be interconnected in a tree like fashion due to the P system's membrane topology. This approach is also explored but it is not the purpose of this paper.

The SNUPS simulator enables parallel execution of all membranes' rules, via threads, but not distribution over a grid or microcontrollers. A distributed simulator for NPS will be available in the next SNUPS release.

4. Membrane controllers for autonomous mobile robots

As stated in section 3, NPSs can be used as a modeling paradigm for distributed control systems for complex systems (mobile robots, for example). In order to prove the efficiency of this approach, two important aspects of the NPSs are explored: the modeling power and the controller performance.

Modeling power. Multiple behaviours are implemented as NPSs in order to prove their modeling power. These membrane systems work both on simulated and real Khepera III and e-puck robots. Only the membrane parameters have to be adjusted in order to work on the two different types of robots. The implemented behaviours which are presented in this paper are: obstacle avoidance, wall following and follow the leader.

Controller performance. One may consider the performance of a controller to be given by the mean execution time of a cycle (see section 4.1) and some behaviour specific indicators. It will be shown that the proposed MCs provide high performances and generate the desired robot behaviours. The benefit is that performance can be increased by improving the simulator's performance (parallelization, distribution and other optimizations) and not necessarily, by modifying the membrane controllers themselves. The simulator (SNUPS) is implemented as a Java servlet and membranes are stored as xml files.

The modeling power is explored in this section, while the performance of the MCs is analyzed in section 5.

4.1. The generic structure of a membrane controller

MCs were developed for the problem of controlling differential wheeled robots, like Khepera III [6] or e-puck [4] robots. These robots are equipped with infrared sensors and DC motors. A general and comparative description of these two robots is presented in Table 1.

The generic structure of all MCs is as follows:

Four main steps can be identified. The first one is the initialization phase of the parameters specific to the robot and to the selected behaviour. In the

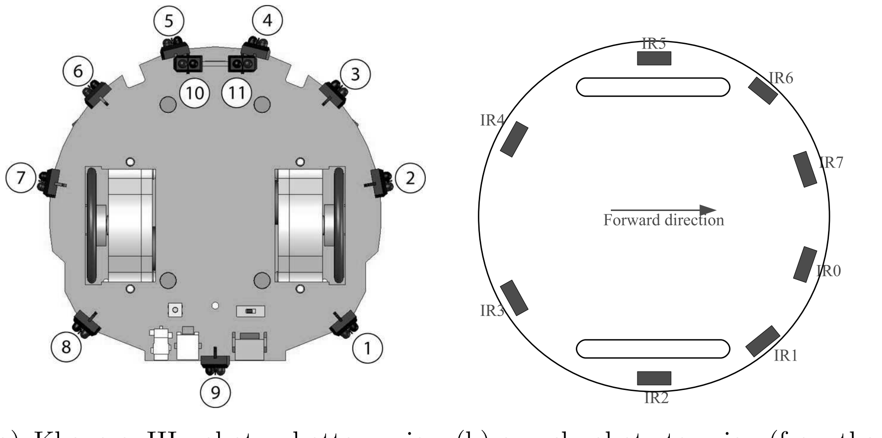
Code 1 The structure of the membrane controller

```
parameters = initializeBehaviourParameters(robot, behaviour)
while(True):
  sensors = readSensors(robot)

  # simulate Numerical P System
  query = constructQuery(robot, behaviour, sensors, parameters)
  response = queryWebApp(address, query)
  speed = extractContent(response)

  setSpeed(robot, speed)
```

second step, the sensors are read. In the third step, the NPS corresponding to the current MC is executed. This is done as an http request. In the fourth step, the motors' speeds are set and the execution of the controller is repeated from step two. The duration of the steps two to four define the execution time of the controller's cycle which is used as a performance measure.



(a) Khepera III robot - bottom view (from the Khepera III manual [6])

(b) e-puck robot - top view (from the e-puck website [4])

Figure 6: IR sensors placement for the Khepera III and e-puck robots

The execution of a NPS is done though a web interface. This interface is basically a Java web application, which receives the values of the input variables of the NPS, calls the SNUPS engine and returns the speeds for the

	Khepera III	e-puck
Processor	dsPIC 30F5011 @ 60MHz	dsPIC 30F6014A
Effectors	2 DC brushed servo motors with incremental encoders	2 stepper motors with incremental encoders, 1 speaker, 1 front led, 1 body led, 1 led ring (8leds)
Max. speed	0.5 m/s	0.129 m/s
Sensors	11 Infra-red (9 around, 2 ground), 5 Ultrasonic	8 Infra-red (around), 1 accelerometer, 3 microphones, 1 CMOS camera
Autonomy	8 hours moving continuously	2-3 hours of intensive use
Comm	RS232 and bluetooth	RS232 and bluetooth
Diameter	130 mm	70 mm
Distance between wheels	90 mm	53 mm
Pulse[mm]	0.047 mm/pulse	0.1288 mm/pulse

Table 1: Khepera III and e-puck hardware description

two motors. To be able to run the web application, a web server and a servlet container must be installed and run on the computer. In order to execute a NPS, a minimum of input parameters must be passed to the web interface. The following parameters must always be defined:

1. *cycleNo* - number of computational cycles to be performed by the NPS;
2. *behaviour* - desired behaviour;

4.2. Obstacle avoidance behaviour

The NPS presented in Fig. 7 is a MC for obstacle avoidance. It implements the following control law:

$$lw = CruiseSpeedLeft + \sum_{i=1}^{8} s_i * weightLeft_i \quad (4)$$

$$rw = CruiseSpeedRight + \sum_{i=1}^{8} s_i * weightRight_i \quad (5)$$

The input variables for the membrane are the values from the infrared sensors, s_i, the weights corresponding to the left and right motors, $weightLeft_i$ and $weightRight_i$, and the cruising speeds of the motors, $cruiseSpeedLeft$ and $cruiseSpeedRight$. The output variables are the speeds of the two motors, lw and rw. Because of the communication constraints of the NPS, the two control laws are computed in sequence. First, the left motor speed is computed and then the right motor speed.

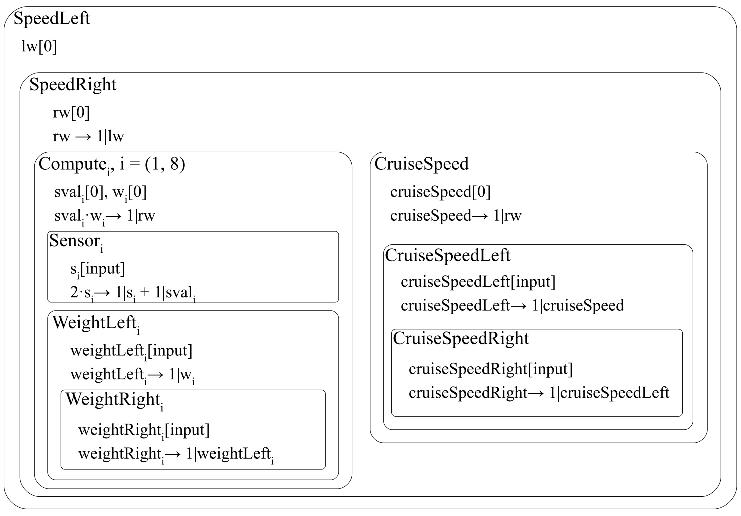


Figure 7: Membrane controller for the obstacle avoidance behaviour

As it is shown in Fig. 7, there are 8 computational membranes, $Compute_i$, and a $CruiseSpeed$ membrane which are used to compute each speed value. The MC executes the following steps:

1. In the first cycle, the following operations are executed:
   (a) sensors' values from the $Sensors_i$ membranes are copied in the $Compute_i$ membranes ($sval_i \leftarrow s_i$);
   (b) the weights corresponding to the left speed are moved in the $Compute_i$ membranes and the weights corresponding to the right speed are moved in their place ($w_i \leftarrow weightLeft_i$, $weightLeft_i \leftarrow weightRight_i$;
   (c) the cruising speed of the left motor is moved to the $CruiseSpeed$ membrane ($cruiseSpeed \leftarrow cruiseSpeedLeft$).

2. In the second cycle:
    (a) the left motor's speed is computed in the variable rw; this value is obtained by summing the contribution from the $Compute_i$ and $CruiseSpeed$ membranes;
    (b) the values (sensor values, weights and cruise speed) used to compute the right motor's speed are moved in the corresponding variables, as described in cycle 1 for the left motor's speed;
3. In the third cycle:
    (a) the left motor's speed is moved in its final place ($lw \leftarrow rw$);
    (b) the right motor's speed is computed in the variable rw; this value is obtained by summing the contribution from the $Compute_i$ and $CruiseSpeed$ membranes;

This MC has to be executed in exactly three cycles in order to return the correctly computed motors' speeds.

The cruise speeds and the weights are dependent on the type of the robot for which the MC is used. Cruise speeds are set such that the robot moves forward with a given speed when there are no visible obstacles. The values of the weights represent the contribution of the sensors to the left, and respectively right motor's speed. For example, the front left sensor has the largest positive contribution to the left motor speed and thus the value of the weight corresponding to that sensor is a positive large number. On the other hand, the same sensor has largest the negative contribution to the right motor's speed and thus the corresponding weight is a large negative number. Based on this, weights have been experimentally determined for the e-puck and Khepera III robots. The scale of the values is determined by the maximum robot's speed and the maximum sensor's reading, while the proportional contribution of each sensor is determined by its placement angle with regard to the front direction of the robot. These values are presented in Table 2. The sensors' values are filtered before being sent to the membrane controller. All values smaller than a given threshold are ignored. It is also important to notice that sensors are numbered differently on Khepera III and e-puck robots as it is shown in Fig. 6.

4.3. Wall following behaviour

The wall following behaviour is implemented as a MC as shown in Fig. 8. The NPS implements the following control law:

	Khepera III	e-puck
SpeedRange	[-43000, 43000]	[-1024, 1024]
MaxIRValue	4095	4095
CruiseSpeed	6000	200
Threshold	100	100
weightLeft	[0, 1, 3, 5, -5, -3, -1, 0, 0]	[-1, -0.6, -0.2, 0, 0, 0.2, 0.6, 1]
weightRight	[0, -1, -3, -5, 5, 3, 1, 0, 0]	[1, 0.6, 0.2, 0, 0, -0.2, -0.6, -1]

Table 2: Khepera III and e-puck obstacle avoidance parameters

$$\begin{aligned} lw &= CruiseSpeedLeft + k_{Dist} * (ref_{Dist} - s_{Dist}) \\ &\quad + k_{Heading} * (ref_{Heading} - (\gamma * s_{Avoid} + s_{FW} - \alpha * s_{BW})) \qquad (6) \\ rw &= CruiseSpeedRight - k_{Dist} * (ref_{Dist} - s_{Dist}) \\ &\quad - k_{Heading} * (ref_{Heading} - (\gamma * s_{Avoid} + s_{FW} - \alpha * s_{BW})) \qquad (7) \end{aligned}$$

The above control laws are responsible for maintaining given distance and heading to the wall. The input variables for the distance to the wall control component are the value of the infrared sensor responsible for measuring the distance to the wall, s_{Dist}, the reference distance to the wall, ref_{Dist}, and the proportional factor for the distance controller, k_{Dist}. The input variables for the heading to the wall control component are the values of the infrared sensors responsible for measuring the deviation from the wall's heading, s_{FW} and s_{BW}, the value of the infrared sensor responsible for detecting a front wall or obstacle, s_{Avoid}, the reference deviation, $ref_{Heading}$ (is set to 0), the proportional factor for the heading controller, $k_{Heading}$, the sensor placement correction factor, α, and the weight of the front sensor, γ. The cruising speeds of the motors are also input variables, $cruiseSpeedLeft$ and $cruiseSpeedRight$. The output variables are the speeds of the two motors, lw and rw.

The NPS presented in Fig. 8 has two membranes, $DistanceController$ and $HeadingController$, which will compute the two control laws (without the cruising speed) in parallel and then these values are added, respectively subtracted, from motor speed values. The MC executes the following steps:

1. In the first cycle, the following operations are executed:
    (a) the distance control component is computed and moved to the main membrane, $WallFollow$, in the variable $uErr$.

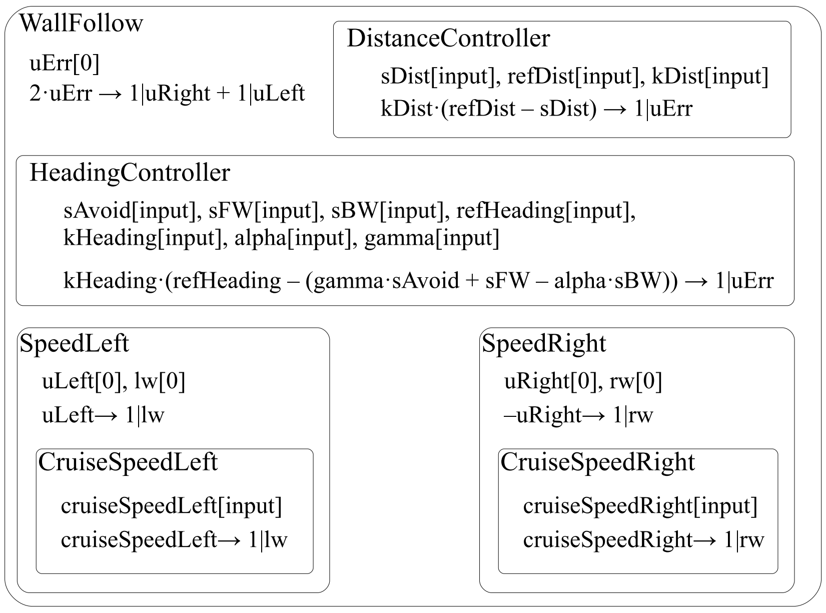

WallFollow
uErr[0]
2·uErr → 1|uRight + 1|uLeft
DistanceController
sDist[input], refDist[input], kDist[input]
kDist·(refDist – sDist) → 1|uErr
HeadingController
sAvoid[input], sFW[input], sBW[input], refHeading[input], kHeading[input], alpha[input], gamma[input]
kHeading·(refHeading – (gamma·sAvoid + sFW – alpha·sBW)) → 1|uErr
SpeedLeft
uLeft[0], lw[0]
uLeft→ 1|lw
CruiseSpeedLeft
cruiseSpeedLeft[input]
cruiseSpeedLeft→ 1|lw
SpeedRight
uRight[0], rw[0]
–uRight→ 1|rw
CruiseSpeedRight
cruiseSpeedRight[input]
cruiseSpeedRight→ 1|rw

Figure 8: Membrane controller for the wall following behaviour

   (b) the heading control component is computed and moved to the main membrane, in variable $uErr$;
   (c) the left cruising speed ($cruiseSpeedLeft$) and the right cruising speed ($cruiseSpeedRight$) are moved to the left and right motor speed variables (lw and rw);
2. In the second cycle:
   (a) the value of the variable $uErr$ is distributed to the two speed membranes, $SpeedLeft$ and $SpeedRight$, in the variables $uLeft$ and $uRight$;
3. In the third cycle:
   (a) in the left motor speed membrane, $SpeedLeft$, the variable $uLeft$ is added to the left motor speed variable, lw;
   (b) in the right motor speed membrane, $SpeedRight$, the variable $uRight$ is subtracted from the right motor speed variable, rw;

In order to compute the motors' speeds, the wall following MC has to be executed for at least three cycles.

The MC from Fig. 8 can be used to follow a wall on the left or right side of the robot with both Khepera III or e-puck robots. This is done by choosing the appropriate set of sensors as input variables for the controller. For example, for the e-puck robot, following a wall on its right side, the third sensor, s_3, has to be selected as the distance measuring sensor (s_{Dist}), the second and the forth sensors as heading deviation measuring sensors (s_{FW} and s_{BW}) and the first sensor as the obstacle detection sensor (s_{Avoid}). The value of the s_{Dist} sensor has to be maintained at the reference ref_{Dist} value. This is done using a proportional law, using the factor k_{Dist}.

The deviation of the robot's heading in respect to the wall is computed as the difference between the values of the two sensors placed on the left and on the right of the normal of the robot's heading. These sensors are the closest to the normal. Because the two sensors are not symmetrical in respect to the normal, a correction factor, α, is used. The difference has to be maintained at zero, thus $ref_{Heading}$ is set to zero. This control is also a proportional one which uses the factor $k_{Heading}$. In order to be able to avoid front walls or obstacles, the weighted value of the front sensor is added as a positive deviation of the heading. Only the value of the front sensor is prefiltered for the wall following behaviour, similar to the avoid behaviour (see Fig. 4.2). In this way the robot can follow convex and concave contours.

The parameters and the set of sensors used in the wall following MC depend on the robot and are shown in Table 3.

	Khepera III	e-puck
s_{Dist}	s_2(left), s_7(right)	s_6(left), s_3(right)
ref_{Dist}	600	300
k_{Dist}	-6(left), 6(right)	-0.5(left), 0.5(right)
s_{Avoid}	s_4(left), s_5(right)	s_8(left), s_1(right)
s_{FW}	s_3(left), s_6(right)	s_7(left), s_2(right)
s_{BW}	s_1(left), s_8(right)	s_5(left), s_4 (right)
$ref_{Heading}$	0	0
$k_{Heading}$	-10(left), 10(right)	-1(left), 1(right)
α	1.25	1.25
γ	0.75	0.75

Table 3: Khepera III and e-puck wall following parameters and sensor sets (see Fig. 6)

4.4. Follower behaviours

In Fig. 9, the MC implementing the 'follow the leader' behaviour is shown. The MC implements the following control law:

$$\begin{aligned} lw &= CruiseSpeedLeft - k_{Dist} * (ref_{Dist} - 0.5 * (s_{Dist1} + s_{Dist2})) \\ &\quad + k_{Heading} * (ref_{Heading} - (s_{R1} + s_{R2} - (s_{L1} + s_{L2}))) \qquad (8) \\ rw &= CruiseSpeedRight - k_{Dist} * (ref_{Dist} - 0.5 * (s_{Dist1} + s_{Dist2})) \\ &\quad - k_{Heading} * (ref_{Heading} - (s_{R1} + s_{R2} - (s_{L1} + s_{L2}))) \qquad (9) \end{aligned}$$

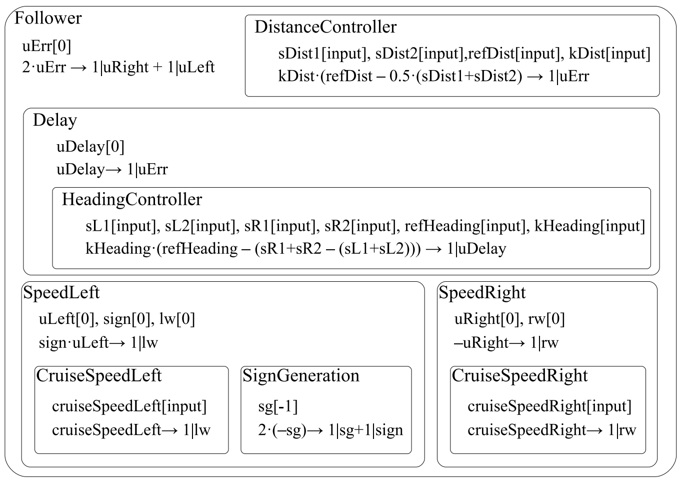


Figure 9: Membrane controller for the follower behaviour

The above control laws are responsible for maintaining given distance and heading to the leader robot. These are very similar to those of the wall following behaviour (see Sec. 4.3). The only differences are the set of sensors for measuring the distance to the leader, s_{Dist1} and s_{Dist2}, and for measuring the deviation of the robot's heading in respect to the leader's heading, s_{R1}, s_{R2}, s_{L1} and s_{L2}. The distance to the leader robot is computed as the mean value of the two front sensors, while the heading deviation is approximated by the difference between the sum of the two left and sum of the two right sensors in respect with the heading of the robot (see Fig. 6). Parameters and sensor sets are presented in Table 4.

	Khepera III	e-puck
s_{Dist1}	s_4	s_1
s_{Dist2}	s_5	s_8
ref_{Dist}	600	200
k_{Dist}	-5	-0.5
s_{R1}	s_5	s_1
s_{R2}	s_6	s_2
s_{L1}	s_4	s_8
s_{L2}	s_3	s_7
$ref_{Heading}$	0	0
$k_{Heading}$	12	0.75

Table 4: Khepera III and e-puck follower behaviour parameters and sensor sets (see Fig. 6)

Another difference is that the control laws are asymmetric, the sign of the components is different for the left and right motor's speeds, as opposed to those of the wall following behaviour. This problem is solved by delaying one of the controller components, in this case the heading component, and alternating the sign of the contribution to the left motor speed, $uLeft$. The delay is implemented using the *Delay* membrane, which wraps the *HeadingController* membrane. The sign of the contribution $uLeft$ is given by the *SignGeneration* membrane, which generates an alternative sign in the *sign* variable of the *SpeedLeft* membrane.

5. Experimental results and discussions

In this section the quantitative results, obtained from experiments on real and simulated robots are presented. These results show that the performance of the MC is sufficiently high in order to achieve a desired robot behaviour. The experiments on real robots (Khepera III and e-puck) were done in an arena with obstacles as shown in Fig. 10. The simulated experiments were done in the Webots 6.2.1 robotics simulator and the arena is shown in Fig. 10(e) and 10(f). For the follower behaviour, the same arenas were used, but without obstacles, because the follower robot cannot distinguish between the leader robot and obstacles by using only the infrared sensors.

All experiments were conducted to cover a variety of interesting cases of the tested behaviours. For example, the wall following experiment was

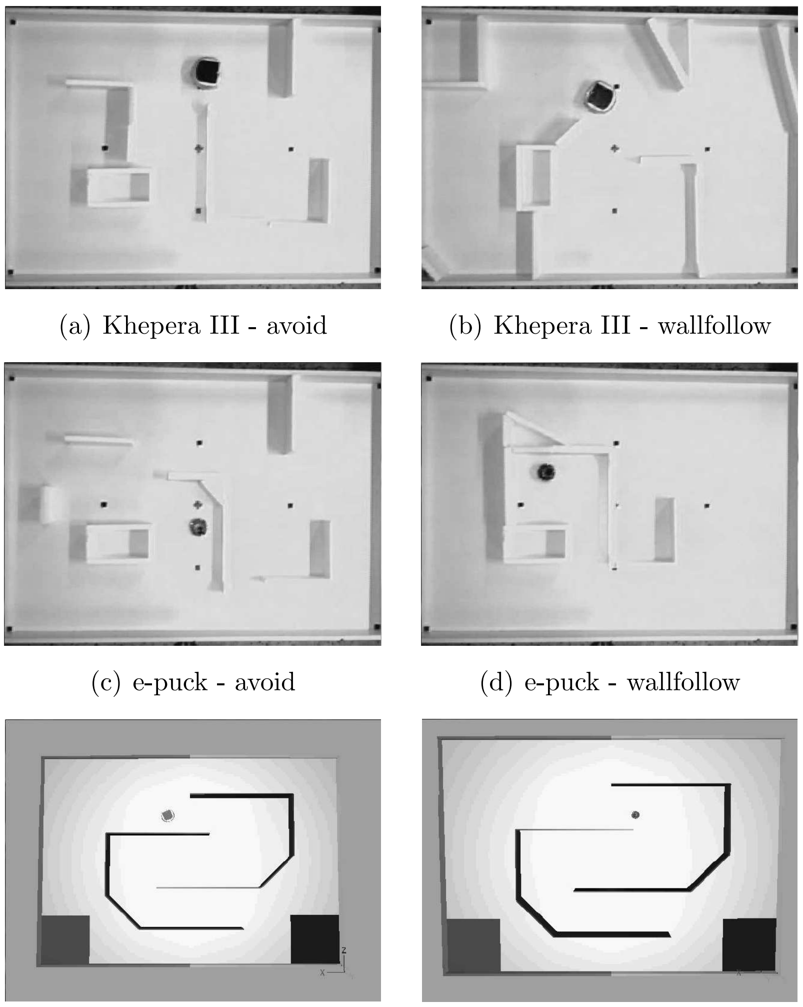

(a) Khepera III - avoid
(b) Khepera III - wallfollow
(c) e-puck - avoid
(d) e-puck - wallfollow
(e) Khepera III - simulation
(f) e-puck - simulation

Figure 10: Arena configurations for the experiments with real and simulated robots

conducted in an arena with walls that contained open and closed corners, straight segments and corners of other angles (Fig. 10(b) and 10(d)). In the obstacle avoidance behaviour, the experiments were conducted in the configurations show in Fig. 10(a) and 10(c) and for an extended amount of time in order to ensure that the robot may encounter as many obstacle avoidance situations as possible.

The common performance index used to evaluate the proposed MCs is the duration of the controller cycle (see section 4). The duration of the cycle includes the communication time with the robot, reading the sensors' values and setting the speed. It is therefore important to measure the execution time of the membrane controller, in this case the http query duration, in order to have a clear perspective of the controller performance. The execution time of the MC (query duration) is stable and very small in comparison to the total cycle duration. The MC is also scalable, the query time does not increase very much with the number of membranes in the controller (the avoid controller has 37 membranes, while the wall follow controller has 7 membranes). The results of the experiments in regard with these performance indices are summarized in table 5 for each behaviour.

The execution time of the MC was chosen as a performance index, because it becomes very important in two cases: 1) when implementing the MC on a system with few resources (memory and CPU power); 2) when the MCs are used as modules in a more complex architecture which must demonstrate a complex cognitive behaviour.

The execution of the membranes was performed on a quad core system under Ubuntu 8.04 for the experiment with real robots and on a dual core system under Windows Vista for the simulations. In all experiments, Apache tomcat 7.04 was used as servlet container. The two configurations determine the difference in the query time (the execution time of the membrane controller) in the two types of experiments. The performance increases, due to the number of available parallel units (processor cores), without modifying the membranes themselves (see table 5).

Behaviour specific performance indices are presented for each membrane controller in the following.

5.1. Obstacle avoidance controller

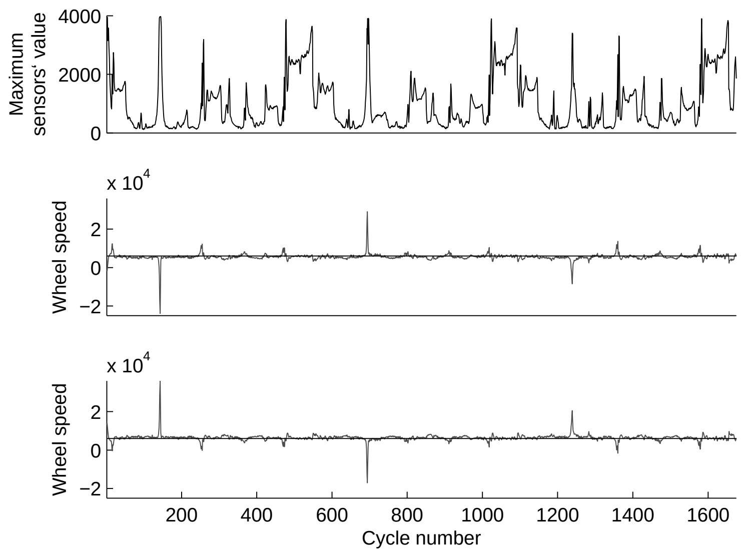
The avoid controller is evaluated based on the maximum value of the sensors. This value is zero when no obstacles are detected and increases when the robot gets close to an obstacle. The evolution of this parameter

Behaviour	Measure		Khepera III		e-puck	
			real	webots	real	webots
Avoid (37 membranes)	cycle time	mean	0.1788	0.0543	0.1815	0.0540
		stddev	0.0136	0.0128	0.0215	0.0114
	query time	mean	0.0131	0.0540	0.0130	0.0538
		stddev	0.0048	0.0127	0.0051	0.0114
Wall follow left (7 membranes)	cycle time	mean	0.1704	0.0276	0.1728	0.0278
		stddev	0.0128	0.0094	0.0187	0.0075
	query time	mean	0.0054	0.0273	0.0052	0.0276
		stddev	0.0032	0.0095	0.0025	0.0075
Wall follow right (7 membranes)	cycle time	mean	0.1709	0.0273	0.1739	0.0276
		stddev	0.0131	0.0065	0.0204	0.0062
	query time	mean	0.0055	0.0271	0.0053	0.0274
		stddev	0.0065	0.0065	0.0025	0.0062
Follower (9 membranes)	cycle time	mean	0.1085	0.0314	0.1745	0.0318
		stddev	0.0189	0.0087	0.0183	0.0123
	query time	mean	0.0066	0.0313	0.0073	0.0316
		stddev	0.0030	0.0088	0.0089	0.0124

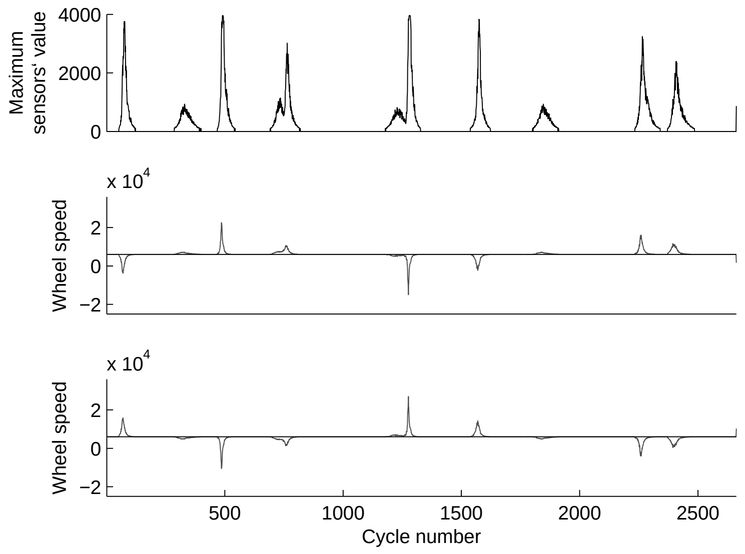
Table 5: Summary of results from experiments with simulated and real Khepera III and e-puck robots (all times in sec)

during the experiments is shown in Fig. 11 and 12. The peaks in the graphs correspond to obstacle detection and avoidance. In the experiments with real robots, this parameter fluctuates more because of the sensors' noise and because the arena is more cluttered than in the simulated experiments (see Fig. 10). The peaks for Khepera III robot are higher than for the e-puck robot due to the fact that Khepera III is not round shaped (see Fig. 6(a)).

In Fig. 11 and 12, are also shown the left and right speeds corresponding to the sensory input. The cruise speed of the robot is represented by the black line as reference. The two speeds equal the cruising speed when the robot does not perceive any obstacle, while the peaks correspond to obstacles that are in the robot's detection area. Details about the evolution of speed profiles are summarized in Table 6, which contains the mean and standard deviation of the speed in absolute value, the maximum and minimum speed and the maximum rate of change.

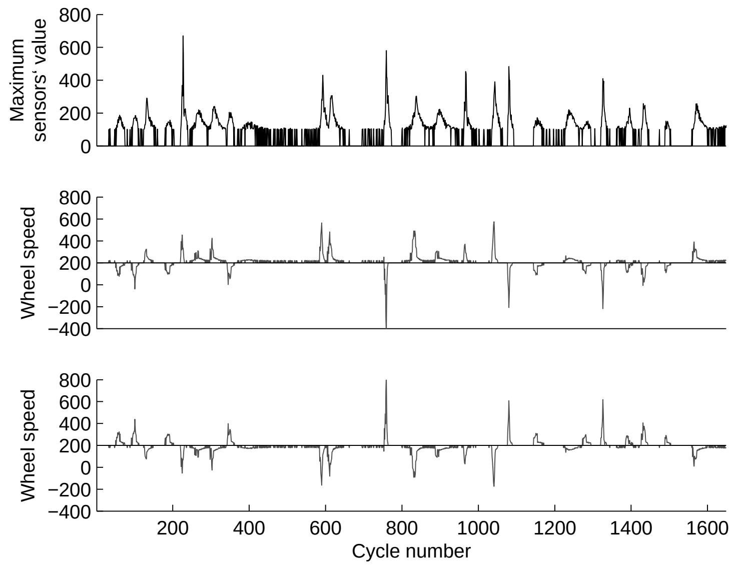


(a) Real Khepera III

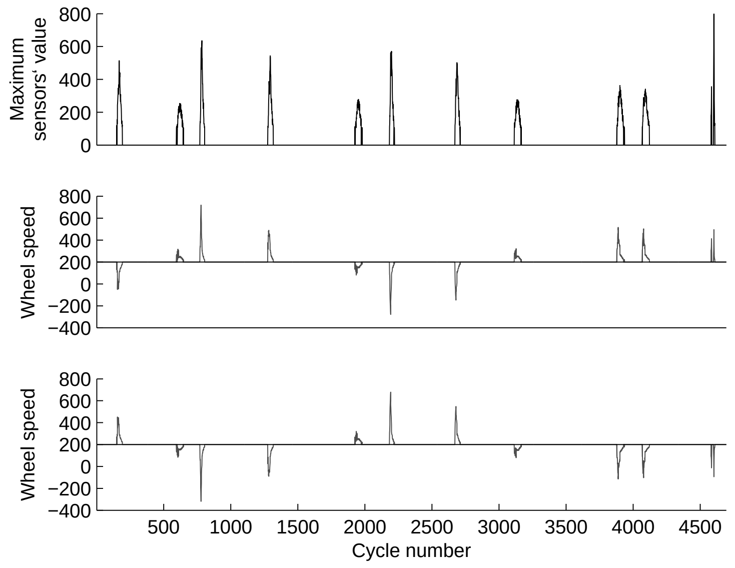


(b) Simulated Khepera III

Figure 11: Evolution of the performance parameter and of speeds for the obstacle avoidance behaviour with Khepera III robots



(a) Real e-puck



(b) Simulated e-puck

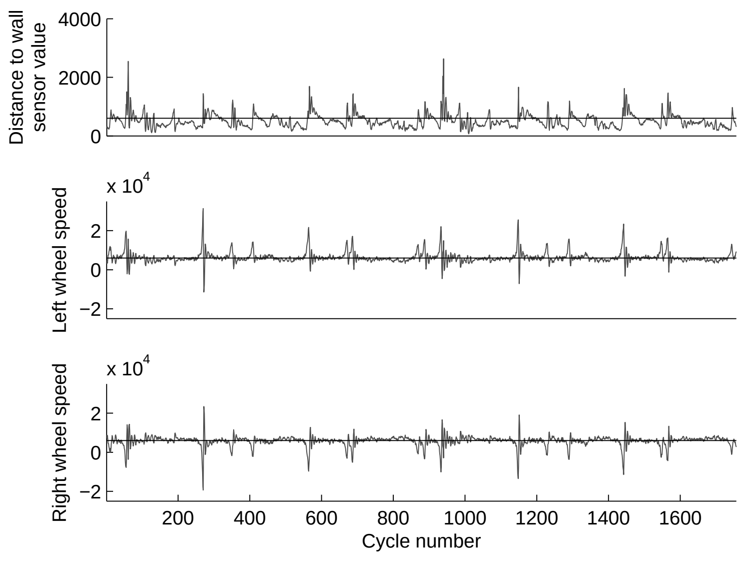
Figure 12: Evolution of the performance parameter and of speeds for the obstacle avoidance behaviour with e-puck robots

Measure		Khepera III		e-puck	
		real	webots	real	webots
distance to objects (sensor raw value)	mean	991	264	82	24
	stddev	888	612	87	83
	max	3974	3962	670	1472
mean speed	left	5650	6158	208	203
	right	6494	5968	195	199
stddev of speed	left	1803	1630	59	42
	right	1803	1630	59	42
maximum speed	left	29059	22554	574	718
	right	35864	26998	815	679
minimum speed	left	-23864	-14998	-415	-279
	right	-17059	-10554	-174	-318
maximum rate of change	left	13878	5842	303	177
	right	13878	5842	303	177

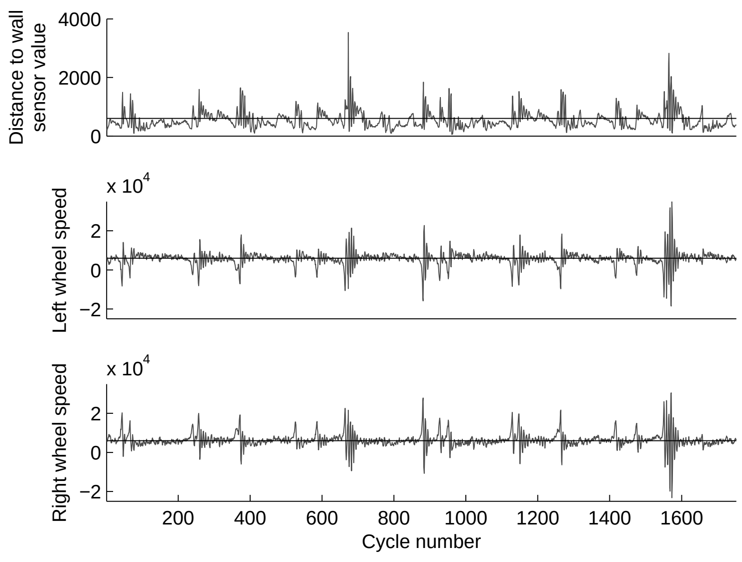
Table 6: Summary of results for the obstacle avoidance behaviour from experiments with simulated and real Khepera III and e-puck robots

5.2. Wall following controller

The performance parameter chosen for the wall follow controller evaluation is the distance to the wall. The distance is measured by an on-board infrared sensor (high values represent small distances, while low values represent large distances). The evolution of the distance to the wall and of the speeds of the two wheels during the experiments with real robots are shown in Fig. 13 and 14. Similar to the obstacle avoidance experiments, the parameter fluctuates more in the experiments on the real robots than in simulation because of the sensors' noise. Peaks in the graphs represent changes of direction caused by corners in the environment. The black horizontal lines represent the reference value for distance sensor, respectively the cruise speed. It can be noticed that the performance parameter and the two speeds are fairly stable during the following of straight segments of the wall. In table 7 there is presented a summary of the experiments with both simulated and real robots. The difference between the reference distance and mean distance to the wall, in absolute value, is show in table 7 as *distance error*. The distance error is smaller in the simulated experiments that it is in the real ones.

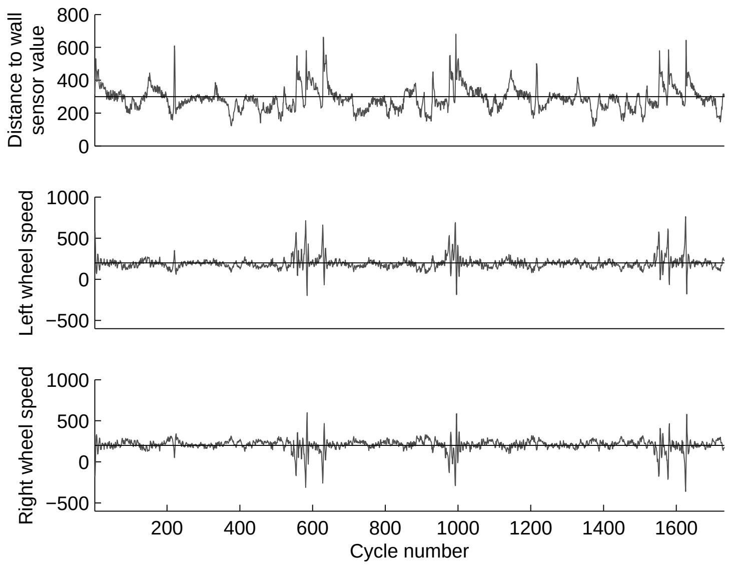


(a) Khepera III - follow wall on the left

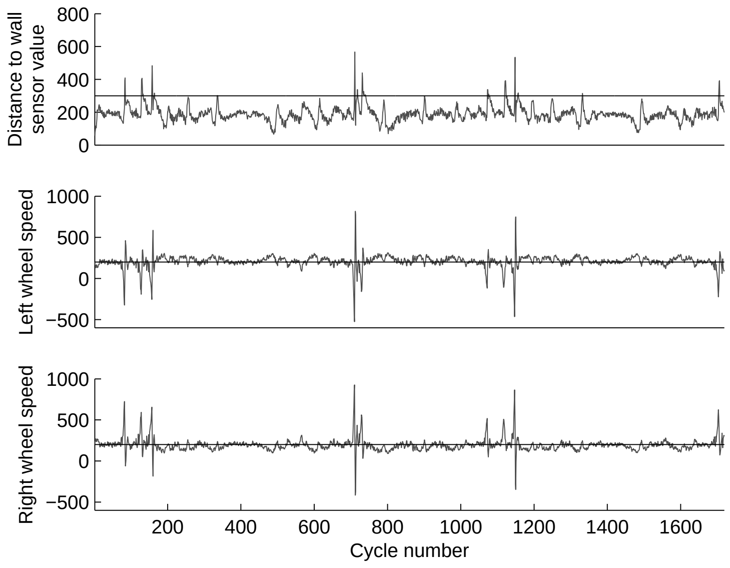


(b) Khepera III - follow wall on the right

Figure 13: Evolution of the performance parameter and of speeds for the wall following behaviour with real Khepera III robots



(a) e-puck - follow wall on the left



(b) e-puck - follow wall on the right

Figure 14: Evolution of the performance parameter and of speeds for the wall following behaviour with real e-puck robots

Side the wall is followed	Measure		Khepera III		e-puck	
			real	webots	real	webots
left	distance to wall (sensor raw value)	mean	514	721	289	264
		stddev	239	354	72	118
	distance error		86	121	11	36
	mean speed	left	6340	5938	194	202
		right	6068	6462	213	213
	stddev of speed	left	2859	3261	74	135
		right	2859	3261	74	135
	maximum speed	left	31336	62708	761	3553
		right	23432	56766	600	1678
	minimum speed	left	-11432	-44766	-200	-1278
		right	-19336	-50708	-361	-3153
	maximum rate of change	left	25984	72750	436	2147
		right	25984	72750	436	2147
right	distance to wall (sensor raw value)	mean	542	604	189	256
		stddev	286	351	47	77
	distance error		58	4	111	44
	mean speed	left	6174	6576	214	210
		right	6536	5936	195	195
	stddev of speed	left	3642	3595	78	93
		right	3642	3595	78	93
	maximum speed	left	35218	57848	817	443
		right	30538	63208	926	3563
	minimum speed	left	-18538	-51208	-526	-3163
		right	-23218	-45848	-417	-43
	maximum rate of change	left	37286	65994	737	2086
		right	37286	65994	737	2086

Table 7: Summary of results for the wall following behaviour from experiments with simulated and real Khepera III and e-puck robots

5.3. Follower Controller

The performance parameter chosen for the follow controller evaluation is the distance to the leader. The distance in this case is set as the mean of the front sensors' values. Fig. 15 shows the evolution of the parameter during experiments. Peaks in these graphs represent corrections in the heading of the robot in order to match the heading of the leader. The leader has a predefined sine motion in order to test speed and heading matching. Table 8 presents the mean and standard deviation of the speeds and distance to the leader.

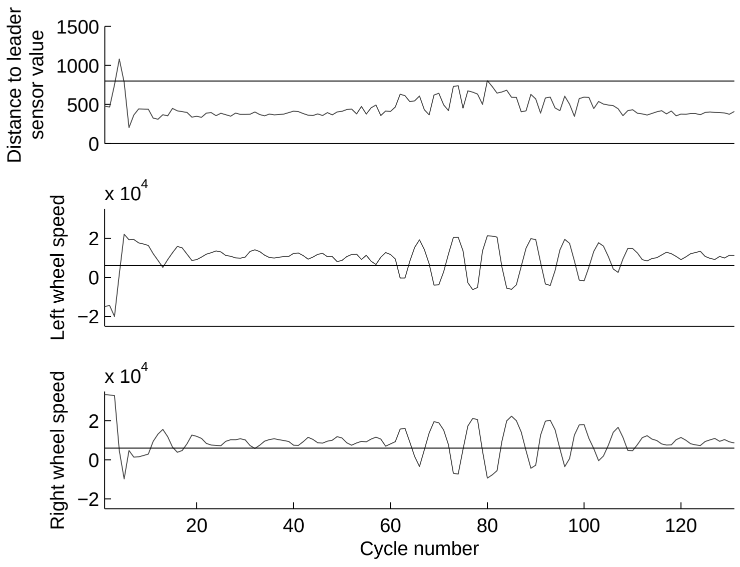
Measure		Khepera III		e-puck	
		real	webots	real	webots
distance to leader (sensor raw value)	mean	459	352	110	136
	stddev	126	314	20	279
mean speed	left	11047	12073	291	367
	right	10210	12773	288	381
stddev of speed	left	7364	13563	42	468
	right	7181	14774	40	536
maximum speed	left	22123	91754	379	1275
	right	33360	90657	374	1296
minimum speed	left	-19991	-81845	155	-4678
	right	-9773	-78850	181	-4645
maximum rate of change	left	21703	92496	113	5065
	right	28313	109984	137	5050

Table 8: Summary of results for the follow the leader behaviour from experiments with simulated and real Khepera III and e-puck robots

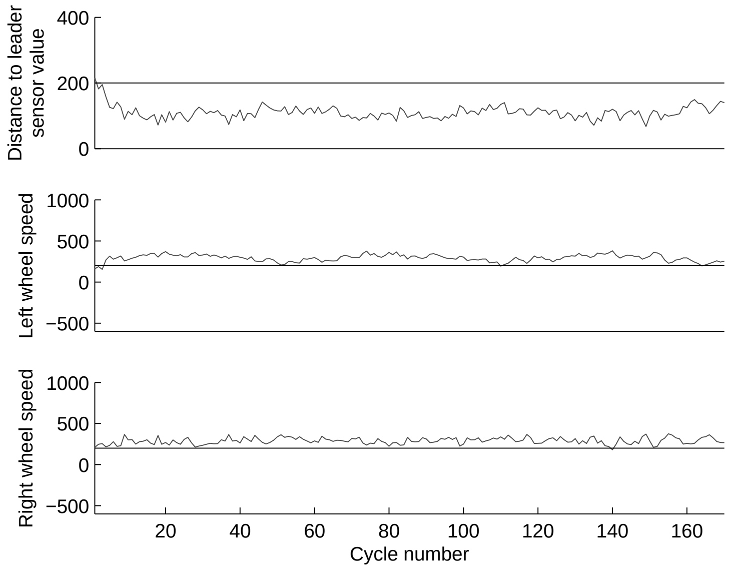
5.4. Fuzzy Avoid Controller

Experiments with a fuzzy logic based obstacle avoidance controller were also performed in order to show that MCs can match the performances of fuzzy controllers. The Mamdani-type fuzzy controller uses sensorial information from the IR sensors of the robot, and has 3 inputs, computed as the sum of the left (2 and 3)), front (4 and 5) and respectively right (6 and 7) sensor group values, see Fig. 6(a). In this way the complexity of the fuzzy controller is reduced. Five Gaussian membership functions were used for each input. This determines a complete rule base of 125 rules. The fuzzy controller was tuned manually in experiments on real robots. The fuzzy controller was implemented and run in Matlab 7.6 under Windows Vista. Fig. 16 presents the maximum sensors' value in each cycle. A summary of the results with the fuzzy obstacle avoidance controller is shown in table 9.

The maximum sensor's value of the two controllers, membrane based and fuzzy logic based, respectively, show that both manage to avoid obstacles



(a) real Khepera III



(b) real e-puck

Figure 15: Evolution of the performance parameter and of speeds for the follow the leader behaviour with real robots

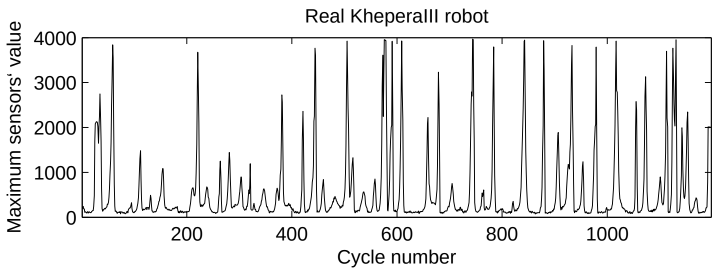


Figure 16: Performance indices in each cycle for the Khepera III robot with fuzzy controller for obstacle avoidance

Measure		Value
distance to objects (sensor raw value)	mean	570
	stddev	810
	max	3966
cycle time	mean	0.2507
	stddev	0.0262
query time	mean	0.0064
	stddev	0.0038

Table 9: Summary of results from experiments with fuzzy obstacle avoidance controller on real Khepera III robots

well. However, there is a qualitative difference in the exhibited behaviours. The fuzzy controller avoids obstacles very sharply, which is why there are so many high, but narrow, peaks in Fig. 16. The peaks in Fig. 11(a) on the other hand are lower and wider. The robot avoids obstacles more smoothly with the MC controller, than with the fuzzy one. This also has as a consequence that the mean value of the performance parameter is smaller for the fuzzy controller than for the MC (Tables 6 and 9).

6. Conclusions and future improvements

This paper introduced the notion of membrane controllers that are built based upon the architecture and functionality of a numerical P system in which variables evolve by means of programs composed of production functions and repartition protocols. The numerical and deterministic nature of this system, together with the distributed and parallel nature and the computing power inherent to P systems, make membrane controllers suitable candidates for the control of complex systems. The proposed innovative concept is validated on experiments on simulated and real mobile robots for which three desired behaviours have been generated (obstacle avoidance, wall following and follow the leader). A comparison with a fuzzy logic controller has been provided.

Further developments will include two main directions. First, more membrane controllers for some other desired behaviours will be generated, possibly by combining desired behaviours (such as follow a leader while avoiding obstacles). Second, the development of symbolic P systems to be used at higher levels in the proposed hierarchy of the cognitive architecture and means to communicate with lower level membrane controllers are to be designed and implemented. More complex robotic applications will be considered, for example collaborative robotics in which teams of robots are asked to jointly achieve a goal, such as identifying and collecting objects.

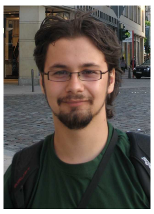
The proposed concept and the software implementation of membrane controllers for robot behaviours opens the way to the development of an integrated biologically inspired cognitive architecture, where the main three modules (execution, coordination, and organization) are membranes themselves, and where there is a combination of symbolic and numerical P systems.

7. Acknowledgements

This work was supported by CNCSIS UEFISCSU, project number PNII IDEI 1692/2008. Kind support of Ana Brandusa Pavel is acknowledged.

8. Biographical information

Catalin Buiu received the B.Sc., M.Sc., and Ph.D. degrees from “Politehnica” University of Bucharest, Bucharest, Romania. He is currently Professor of Cognitive Robotics and Head of the Natural Computing and Robotics Laboratory at the same University. His research interests include cognitive robotics, natural computing, mathematical modeling of biological processes, and educational technologies.



Cristian I. Vasile received the B.Sc. from the “Politehnica” University of Bucharest, Romania. He is a Ph. D. student and member of the Natural Computing and Robotics Laboratory of the same University. His primary research interest are collaborative robotics, swarm intelligence, human-swarm user interfaces and distributed algorithms.

Octavian Arsene received the B.Sc. from Military Technical Academy, M.Sc. from Military Technical Academy and “Politehnica” University of Bucharest, Romania. He is now a Ph.D. student with the Department of Automatic Control and Systems Enginnering from “Politehnica” University of Bucharest. He is currently a Senior Software Engineer with Oracle Romania and scientific researcher at Biomedical Engineering Center at “Politehnica” University of Bucharest. His research interests include cognitive architectures, software agents, bio-inspired computing.

References

[1] C. Buiu. Towards integrated biologically inspired cognitive architectures, keynote talk. In *Proc. of the Int. Conf. on Electronics, Computers, and AI - ECAI' 09, Pitesti, Romania*, I, pages 2–8, 2009.

[2] C. Buiu, O. Arsene, C. Cipu, and M. Patrascu. A software tool for modeling and simulation of numerical P systems. *BioSystems*, 103 (3): 442–447, 2011.

[3] DARPA. *DARPA's call for biologically inspired cognitive architectures research program.* www.darpa.mil/ipto/programs/bica/bica.asp, 2010.

[4] epucksite. *e-puck website.* www.e-puck.org, 2010.

[5] L. Huang, I. H. Suh, and A. Abraham. Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants. *Information Sciences*, 181(11):2370 – 2391, 2011.

[6] F. Lambercy and G. Caprari. *Khepera III manual ver 2.2.* http://ftp.k-team.com/KheperaIII/Kh3.Robot.UserManual.2.2.pdf, 2010.

[7] H. Lodish. *Molecular Cell Biology.* Freeman, 5th edition, 2003.

[8] G. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.

[9] G. Paun. *Membrane Computing - An Introduction.* Springer-Verlag, Berlin, 2002.

[10] G. Paun and A. Paun. Membrane Computing and Economics: Numerical P Systems. *Fundamenta Informaticae*, pages 213–227, 2004.

[11] G. Paun and G. Rozenberg. A Guide to Membrane Computing. *Theoretical Computer Science*, 287 (1):73–100, 2002.

[12] G. Paun, M. J. Perez-Jimenez, and G. Stefanescu. Membrane computing and programming. *Journal of Logic and Algebraic Programming*, 79(6): 289 – 290, 2010.

[13] G. Paun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.

[14] A. Pavel, O. Arsene, and C. Buiu. Enzymatic numerical P systems - a new class of membrane computing systems. In *The IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010) Liverpool*, pages 1331–1336, September 2010.

[15] snups. *SNUPS home page.* snups.ics.pub.ro, 2010.

[16] J. Zhao and N. Wang. A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling. *Computers and Chemical Engineering*, 35(2):272 – 283, 2011.