Special Issue

Simulation: Transactions of the Society for Modeling and Simulation International
1–13
© The Author(s) 2024
DOI: 10.1177/00375497241292347
journals.sagepub.com/home/sim

A test-driven methodology for designing robot controllers and simulators using enzymatic numerical P system models

Radu Traian Bobe[1], Marian Gheorghe[2], Florentin Ipate[1] and Ionuţ Mihai Niculescu[1]

Abstract
The design of model-based robot controllers or simulators requires the use of adequate approaches for selecting the models that fit their functions and intended behavior. In this paper, we propose a methodology based on a test-driven method for designing robot controller simulators that use enzymatic numerical P system models. By applying this methodology, different instances of the model are obtained by tuning different aspects, such as the structure, functions, and parameters of the model based on a set of testing scenarios. Another set of tests is used to validate the model instances obtained. The paper, through the methodology proposed, provides an effective way of combining modeling, simulation, and testing in conjunction with a set of tools associated with them.

Keywords
Membrane computing, numerical P systems, enzymatic numerical P systems, robot controllers, simulation, search-based software testing

1. Introduction

In recent years, the evolution of *membrane computing* in the realms of computer science, economics, and biology has sparked interest and innovation. Defined as a branch of an area that investigates computational models inspired by the nature processes, called *natural computing*, *membrane computing* was introduced by Păun[1] over 20 years ago. This biological-inspired computational paradigm investigates models that derive from the structure of a living cell. As the results were promising, the significant milestones of the first decade of research in this field were presented in Păun et al.[2] Moreover, the main classes of a membrane systems were involved in real-life scenarios.[3]

Different variants of P systems have been proposed, such as tissue P systems,[4,5] spiking neural P systems[6–11] or P systems with active membranes.[12,13]

Numerical P systems have been introduced with the aim of modeling economics phenomena, in a nature-inspired computational setting.[14] Later on, an extension of this model, called *enzymatic numerical P system* (EN P system),[15] was introduced as being adequate for modeling robot controllers.[16–18] PeP,[19] a simulator for running EN P systems, has been provided and utilized in various controllers.[16,17,20]

Software testing is an essential part of the development process of applications, serving as a fundamental mechanism for ensuring their functionality, safety, and compliance to requirements. As software applications tend to become indispensable in solving real-life problems, testing plays a pivotal role in identifying undesired behavior. An approach to explore the complex search spaces inherent in modern software systems and discover the unsatisfactory functionalities is search-based testing.[21,22] Search-based testing has expanded its applicability in autonomous systems and a relevant tool for generating tests in this area is AmbieGen.[23]

In this paper, we propose a methodology for building simulators for a robot controller based on EN P system models. Model instances are built using the formalism provided in Pavel et al.[15] The development process of

[1]Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Romania
[2]Faculty of Engineering and Informatics, University of Bradford, UK

Corresponding author:
Radu Traian Bobe, Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Str Academiei 14, 010014 Bucharest, Romania.
Email: radu.bobe@s.unibuc.ro

designing the model instances is dictated by a number of scenarios aimed at highlighting the behavior of each model version. In some sense, our approach is similar to "test-driven development,"[24] improvements being added only after studying the performance of the models during the previous tests. The behavior of the controller designed for an educational robot, called *E-puck*,[25] is analyzed in a virtual environment using Webots,[26] a robot simulation tool.

This paper is an extended version of the article[27] presented at EAI SIMUtools 2023 conference.[28] Compared to the initial article, we introduced and applied a methodology to determine the sensor parameter values (*weights*) based on a linear regression algorithm. This heuristic led us to develop a new model which is presented in detail in this extended version. We also extended the simulation phase, adding an experiment consisting of 25 test cases, aimed at proving the validity of the newest version of the model, compared to the previous ones.

The paper is structured as follows: The "Preliminaries" section introduces the definition of EN P systems along with the fundamentals of regression and search-based test generation applied in robotics. The "Methodology Description and Research Environment" section presents the working methodology, explaining each step of our experimental approach. The "Enzymatic Numerical P System Models" section describes different instances of EN P system model that we designed, while the "Simulation Results" section illustrates the simulation experiments and presents the results in a comparative manner. In the end, the "Conclusions and Future Work" section states the conclusions and future work.

2. Preliminaries

2.1. EN P system definition

In order to facilitate understanding of the methodology expounded in this paper, along with its operational framework, we find it useful to introduce the EN P system definition from the original article,[27] along with the first three instances of the model, presented in the "Enzymatic Numerical P System Models" section.

The EN P systems are special classes of membrane systems, that share with the rest of the models only the membrane structure, in the form of a tree. The compartments contain *variables* instead of objects and their values are processed by *programs* replacing the rewriting and communication rules.[15] As in any membrane system, the compartments are delimited by membranes. Subsequently, we use them interchangeably. A global clock controls the systems through discrete time units.

The EN P system is defined by the tuple:

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

where

- $m \geqslant 1$ is degree of the system Π (the number of membranes);
- H is an alphabet of labels;
- μ is membrane structure (a tree);
- Var_i is a set of variables from membrane $i, 1 \leqslant i \leqslant m$;
- $Var_i(0)$ is the initial values of the variables from region $i, 1 \leqslant i \leqslant m$;
- Pr_i is the set of programs from membrane $i, 1 \leqslant i \leqslant m$.
- The program $Pr_{l_i,i}$, $1 \leqslant l_i \leqslant m_i$ has one of the following forms:

Non-enzymatic:

$$F_{l_i,i}(x_{1,i}, \dots, x_{k,i}) \rightarrow c_{1,i}|v_1 + c_{2,i}|v_2 + \\ \dots + c_{m_i,i}|v_{m_i} \tag{2}$$

where $F_{l_i,i}(x_{1,i}, \dots, x_{k,i})$ is the production function, $c_{1,i}|v_1 + c_{2,i}|v_2 + \dots + c_{m_i,i}|v_{m_i}$ is the repartition protocol, and $x_{1,i}, \dots, x_{k,i}$ are variables from Var_i. Variables $v_1, v_2 \dots v_{m_i}$ belong to the compartment where the programs are located, and to its upper and inner compartments, for a particular compartment i.

Remark 1. *If a compartment contains more than one program, only one will be non-deterministically chosen.*

2. Enzymatic:

$$F_{l_i,i}(x_{1,i}, \dots, x_{k,i})|_{e_i} \rightarrow c_{1,i}|v_1 + c_{2,i}|v_2 + \\ \dots + c_{m_i,i}|v_{m_i} \tag{3}$$

where e_i is an enzymatic variable from Var_i, $e_i \notin \{x_{1,i}, \dots, x_{k,i}, v_1, \dots, v_{m_i}\}$. The program can be applied at time t only if:

$$e_i > min(x_{1,i}(t), \dots, x_{k,i}(t)) \tag{4}$$

Remark 2. *The system evolves in time by executing all the programs that meet condition 4 (the enzymatic control). This happens in parallel in all compartments.*

When the program is applied by the system at time $t \geqslant 0$, the computed value is as follows:

$$q_{l_i,i}(t) = \frac{F_{l_i,i}(x_{1,i}(t), \dots, x_{k,i}(t))}{\sum_{j=1}^{n_i} c_{j,i}} \tag{5}$$

representing the *unitary portion* that will be distributed to the variables $v_1, \dots, v_n$, proportional to coefficients $c_{1,i}, \dots, c_{m_i,i}$, where $c_{j,i} \in \mathbf{N}^+$ and the received values will be $q_{l_i,i}(t) \cdot c_{1,i}, \dots, q_{l_i,i}(t) \cdot c_{m_i,i}$.

Remark 3. *The value of each of the variables from time $t-1$ occurring in the production functions is consumed (reset to zero) and its new value is the sum of the proportions distributed to variable through the repartition protocols, if it appears in them, or remain at value zero if not.*

2.2. Regression applied in robotics

The interdisciplinary nature of the robotics field is undoubted considering the need of creation, design, and use to automate tasks of increasingly complex difficulty, designed to simplify people's daily lives. Industries like automotive, healthcare, logistics, or agriculture are just a few of areas where the use of robots has led to the optimization of certain processes and increased their safety. Due to the endowing with sensors, actuators, and computational systems, robots are capable to perceive and interpret environmental conditions, making decisions and guiding themselves in real time. All these actions are possible by interconnecting the robot components through the core element, called the robot controller. The robot controller is responsible for interpreting external influences and making decisions based on the information provided by the sensors. The importance of having a robust implementation of controller is all the greater as robotics revolutionizes industries by boosting the safety of the systems.

In this paper, we discuss the use of a robot controller based on EN P systems designed to avoid obstacles and following roads of different complexities. In fact, passing a curved road can be treated as an extension to an obstacle avoidance scenario, since the road borders can be interpreted by the proximity sensors of the respective robot as obstacles.

As mentioned above, designing the robot controller is an essential step in order to obtain the expected behavior. In our initial approach,[29] we used a model that associates proximity sensor values with constant parameters (called weights) that were empirically chosen in order to obtain the desired performance. In our current approach, we propose a parametrization based on linear regression. Before getting into the details, let us introduce some fundamental concepts regarding regression.

In a regression problem,[30] the objective is to discover a function that efficiently models a hidden function that represents the relationship between input and continuous output, trying to obtain the closest possible approximation. The goal can be also interpreted as predicting the dependent variables based on independent variable values.[31] Simple linear regression is a regression model with a single independent value, while multivariate linear regression (MLR)[32] predicts the result of an answer variable, using a number of independent variables. The expression that describes this definition is the following:

$$y = \beta_0 + \beta_1 \cdot x_1 + \cdots + \beta_m \cdot x_m + \epsilon \qquad (6)$$

where y is the dependent variable and x_i, $1 \leqslant i \leqslant m$, the independent variables. β parameters are fixed real numbers (weights), while ϵ is the random error term, with a mean of 0 and constant variance, used to express the effect of random factors on dependent variable.

Regression is used in two different areas. First, regression analyses are usually applied for prediction and forecasting. The main types of supervised learning problems[33] include regression and classification problems, the major difference between them being the discrete or continuous character of the output. Alternatively, regression can be used to determine relations between the independent and dependent variables. This analytical technique is widely used in fields such as statistics and economics,[34] but has considerable applications even in robotics, as the information extracted from regression analysis can contribute to decision-making processes.

Upon closer examination, considering the implications of regression in the field of robotics, we can observe significant progress. For example, Chavez-Olivares et al.[35] presented seven identification schemes based on different regression models, as a procedure to obtain the identification of the parameters of a robot manipulator. Furthermore, as the performance of robots in performing certain tasks increases, the internal design of robots grows in complexity. A survey paper regarding the use of regression algorithms in the context of learning mechanical models of robots has been published by Olivier Sigaud, Camille Salaün, and Vincent Padois.[36] This topic is also presented in depth by D. Nguyen-Tuong and J. Peters.[37] Analyzing all the presented methods, it can be concluded that modeling robot systems using regression is still a challenge, improvements in performance of the algorithms still needed to be done. The dynamic behavior and core capabilities of a robot are characterized by the process of parametrization. In robotics, it consists of specifying the parameters that govern the functionalities of a robot. This approach is exemplified in estimating the parameters of an industrial robot for controller design purposes.[38] Lizana et al.[39] issued different parametrization methods of phase change modeling, including static linear transition using linear regression.

In the approach outlined in the present work, we employ linear regression to identify appropriate values for the proximity sensors' weights of a robot controller. As these constant values were chosen in an empirical manner in the previous work, we aimed to determine the relationship between one dependent value (the speed of the wheel) and six independent values (proximity sensors' values). In-depth details regarding the implementation and working methodology will be elucidated in the subsequent dedicated section.

2.3. Search-based test generation in robotics

Robust testing is critical in the realm of software-controlled complex systems, such as autonomous vehicles, smart grids, or healthcare monitoring systems, ensuring reliability in interlinked ecosystems. The rapid development of this field has led to the discovery of innovative methods for predicting behavior[40;] however, the complexity of the scenarios that must be considered remains high. As the diversity of cyber-physical systems increases, modern and efficient testing approaches are needed. An important innovative method to achieve this was *model-based testing*, where the test specification is derived from both the system requirements and a model that describes selected functional and non-functional aspects of the system under test.[41,42]

Search-based software testing[43] has contributed significantly to overcome common testing problems, using metaheuristic optimizing search techniques, such as genetic algorithms.[44] Search-based test generation has been applied in virtual scenarios to test cyber-physical systems, whose testing is not feasible to be done in a real environment. Arrieta et al.[45] proposed a search-based approach for cyber-physical systems' test generation, implemented on top of Non-dominated Sorting Genetic Algorithm II (NSGA-II), a commonly applied multi-objective algorithm. Search-based software testing can also be integrated with complementary techniques, with a pertinent example being the work of Togelius et al.,[46] who combined it with procedural generation. Gambi et al.[47] used search-based software testing combined with procedural generation for testing self-driving car software.

Resuming our focus on testing cyber-physical systems, particularly within the domain of robotics, virtual test generation using different methods is essential in order to create different challenging scenarios for the robot.[48] In this paper, we used roads tests generated with a search-based tool that will be detailed in the next section, having diverse levels of difficulty, dictated by the number and contour characteristics of the curves.

3. Methodology description and research environment

This section presents the methodology behind our approach, its key components, and main functions. The tools supporting our approach are introduced and their roles in implementing the methodology described.

Now that we have introduced the tools that make up the research environment, we can examine the methodology used in our experiments. First, the scope of this paper is to build simulators for a robot controller based on EN P systems model.

We used road spines generated with AmbieGen to create roads with a specified width. In addition to the tool-generated roads, we also used manually created tests, such as circular arena with obstacles, corridors, and a square, created in Webots.

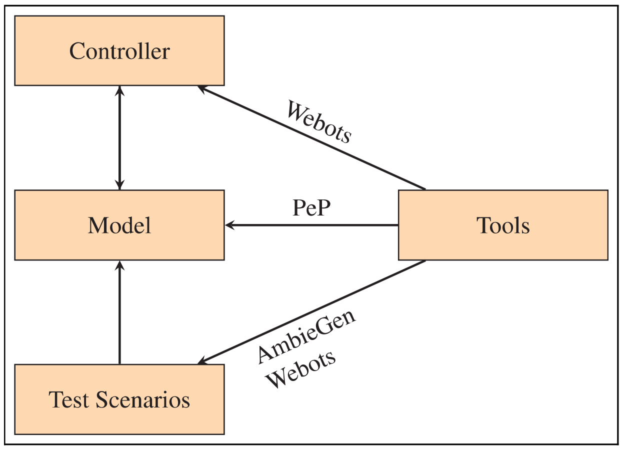

Figure 1. Flow diagram.

The *methodology* utilized in our approach is based on some key *components* and *interactions* among them. A visual representation of the main aspects of our proposed methodology is presented in Figure 1.

The components are: a *robot controller*, a *formal model*, which is the core part of the controller, and *test scenarios* aimed to check the functionality of the controller and validate its behavior. We rely on a *test-driven design* for the methodology, whereby the test sets guide the process of building the controller, adapting accordingly the formal model.

Now, we describe the functions of each component, also revealing the interactions among the components.

The *robot controller* is the core component of the methodology. This controller is meant to equip an E-puck robot.[25] An important aspect of our study is the fact that the robot has two motors attached to the body along with two wheels; the speed value is changeable and handled by the controller. It also includes eight infrared proximity sensors placed around the body and a GPS attached to the turret slot in order to assess the coordinates. The controller takes the proximity sensor values from the robot and, using the formal model, transforms them into velocities for both motors of the robot. To gain a clearer understanding of how the controller operates with the mentioned data, next we present the pseudocode version of the main loop of robot controller, defined by us in our previous work.[29]

Instead of using the controller directly on the E-puck robot, we illustrate its behavior and also the methodology, by using a simulator, built with Webots, a robot simulation tool, allowing to construct complex environments.[26]

The controller's backbone is a formal model, in the form of an *EN P system*. As the controller design, and implicitly the model, is guided by test scenarios, their structure and functionality may change as a consequence of adjustments resulting from the results of these tests. The

Algorithm 1 Simulation steps performing algorithm

```
1: repeat
2:   for i = 1 to number of sensors do
3:     sensor membrane(i) ← value(i)
4:   run one simulation step
5:   read lw, rw from P system
6:   leftMotor ← lw
7:   rightMotor ← rw
8: until the end of the road or E-puck goes out of the road
```

model and the controller may evolve. In our case, we have obtained a total of five models—their precise definitions appear in the next section, but here we only mention that we start with a model introduced by Florea and Buiu.[20] Motivated by the aim to obtain not only favorable test outcomes, but also the most natural behavior of the robot, a new model is refined from the previous model. It is important to mention that evolving a model may involve changing its structure, functionality, by changing the rules, and parameters—all these are presented in detail in the next section.

EN P system models are simulated (executed) with a tool called PeP,[19] allowing to define them in a domain-specific language and then execute the systems in accordance with the semantics of the models. The robot controller is then integrated with the Webots simulator.

Below we describe the way the parameters are adjusted, the changes to structure and functionality will be presented when the models will be introduced in the next section. Before presenting the parameters' adjustments, we present the test scenarios.

To strengthen the testing phase, which in our methodology plays a crucial role not only in validation but also in defining new model variants, we used different test cases in two stages: first, to observe the limitation of each previous model instance and based on the results to design a new one and second, to validate the solutions obtained after the first stage. Test scenarios are divided into areas the E-puck robot is meant to go through and roads of different types and sizes. For the first set of test scenarios, different Webots visual objects (shapes)[26] are used which are populated with obstacles, by using a special mechanism, called PROTO,[49] that allows to build special objects within an environment. Roads generation is achieved by using AmbieGen, an open-source tool relying on evolutionary search methods for the generation of test scenarios.[23,50] The software is written in Python and uses evolutionary methods[51] and multi-objective algorithms for search-based test generation.[44] A complete description of these test scenarios and the results of the validation process are presented in the "Simulation Results" section.

In the initial phase, each sensor value had a constant parameter, called *weight*, which was empirically chosen to obtain the desired conduct. After refining the initial model, obtaining new models with more accurate behavior, our last model uses a parametrization method in order to determine a value for these weights. To achieve this, we used a linear regression algorithm implemented in Python.

As mentioned in the "Regression applied in robotics" subsection, we aimed to determine the relationship between the speed of the wheel and proximity sensors' values, using a linear regression with one dependent value (the speed) and six independent values(sensors values). Prior to the application of the algorithm, relevant data collection was needed. To accomplish this, we used an obstacle avoidance controller[52] presented in Webots documentation web page. This controller gets the sensors' values in a similar way to our controller, but detects if a collision occurs using the value returned by a distance sensor and comparing it to a threshold, defined as a constant float value. Finally, the information about collision detection is used to actuate the wheels, modifying speed according to obstacles.

Understanding the operational mode of this controller, we applied it on our test scenarios, taking at each step of the simulation the sensor values as well as the actual speed of the motors. It is important to mention that we used just six out of the eight proximity sensors that E-puck is equipped with, because we needed just the data from the sensors placed radially at the frontal part of the robot.

Relevant data were extracted from the circular arena with obstacles, as all the sensors were triggered during the experiment. The data collected were then introduced in Python, in the linear regression script, obtaining parameter equal in values, but of opposite sign, for left and right wheel.

Using the weight values obtained in the above presented way, we designed our last model, entitled in the next section as *Linear Regression-Based Model*. The model exhibits a visibly improved structure compared to the previous one, being much simpler and easier to understand, utilizing the parameter values returned by regression and not empirical values.

Now, we briefly describe how the tools mentioned so far are integrated:

- PeP simulator contains the EN P system model
- The model is integrated with the E-puck controller via PeP
- AmbieGen provides the test generation scenarios with roads as *.json* files, plus test outcome, maximum curvature coefficient, and so on.
- In the Webots graphic interface, the simulation can be visualized on inputs provided by AmbieGen. More details about the tools and the way they work together are available from Bobe et al.[29]

Remark 4. *All the models used in this experiment and test scenarios along with the results are available in our GitHub project.*[53]

4. EN P system models

This section presents our different instances for the model used to control the robot. As the linear regression-based model is created following an analysis of experimental results and the drawbacks of the previous model structure, we proceed to introduce in the beginning the way we obtained the instances of the model as well as the prior three versions, as extracted from the conference article.[27]

The controller receives data from proximity sensors, that measure distances to obstacles from the environment, in order to determine the direction of movement of a differential two-wheeled robot, E-puck, in our case.

The proximity sensor has a range of 4 cm; if the obstacles are further than this limit the sensor returns the value of 0. The proximity sensors are placed on the left and right side of the robot in the direction of its movement at different angles.

The basic model was taken from[20] and adapted to make a rotation move when an obstacle is near, in order to avoid it and continue the movement. The equations that calculate the linear and angular velocity are shown below:

$$leftSpeed = cruiseSpeed + \sum_{i=1}^{n} weightLeft_i \cdot prox_i \quad (7)$$

$$rightSpeed = cruiseSpeed + \sum_{i=1}^{n} weightRight_i \cdot prox_i \quad (8)$$

The *leftSpeed* and *rightSpeed* are the speeds of the two wheels of the robot, while *n* is the number of sensors. Each sensor has constant weight values, empirically chosen to conduct the robot to the desired behavior. This basic model encountered considerable difficulties to pass the road tests generated by AmbieGen, being capable to move without hitting the margins just on straight roads or very little curved roads. The model was formally described and more experiments have been done in our previous work,[29] where it is referred as Π_{M_1}.

Considering the limitations of Π_{M_1}, the model we present next is an improvement on the first one.

4.1. Basic model with rotation

Analyzing the initial model, we observed that the speed of the wheel on the side with an obstacle increased reported to weights. Thereby, this caused a sort of rotation in the opposite direction of the obstacle but not sufficient to pass the test.

The main change of our model was to introduce the compartment *w*, calculating the product of the travel speed and the sum of the weights. In this way, after rotating in the opposite direction when detecting an obstacle, the robot will continue the test, moving forward with a constant velocity. This is certainly an improvement, especially when the robot is challenged on roads, but as presented next, there were still some issues that aimed us to introduce another model.

Let us consider the following function:

$$f(x) = \begin{cases} 1, & if\ x = 0 \\ 0, & otherwise \end{cases} \quad (9)$$

This function will be used in the equations describing the behavior of the model and in the production functions from the programs.

The equations describing the behavior are as follows:

$$weightLeft = \sum_{i=1}^{n} weightLeft_i \cdot prox_i \quad (10)$$

$$weightRight = \sum_{i=1}^{n} weightRight_i \cdot prox_i \quad (11)$$

$$leftSpeed = cruiseSpeed \cdot weightLeft + f(weightLeft) \cdot cruiseSpeed \quad (12)$$

$$rightSpeed = cruiseSpeed \cdot weightRight + f(weightRight) \cdot cruiseSpeed \quad (13)$$

The model is defined as follows:

$$\Pi_{M_2} = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0))) \quad (14)$$

where

- $m = 3k + 3, k = 6;$
- $H = \{s, w, s_c\} \cup \bigcup_{i=1}^{k} \{c_i, s_i, w_i\};$
- $\mu = [[[[]_{s_1} []_{w_1}]_{c_1} \ldots [[]_{s_k} []_{w_k}]_{c_k} []_{s_c}]_w]_s;$
- $Var_s = \{x_{s_l}, x_{s_r}\}, Var_w = \{x_{w_l}, x_{w_r}, e_w\},$
  $Var_{s_c} = \{x_{s_c}\},\ Var_{c_i} = \{x_{c_i,s_l}, x_{c_i,s_r}, x_{c_i,w_l}, x_{c_i,w_r}, e_{c_i}\},$
  $1 \leqslant i \leqslant k,$
  $Var_{s_i} = \{x_{s_i,i}\}, 1 \leqslant i \leqslant k,$
  $Var_{w_i} = \{x_{w_i,w_l}, x_{w_i,w_r}, e_{w_i}\}, 1 \leqslant i \leqslant k;$
- $Var_i(0) = 0, 1 \leqslant i \leqslant m;$
- $Pr_s = \{0 \cdot x_{s_l} \cdot x_{s_r} \rightarrow 1|x_{s_l} + 1|x_{s_r}\};$
  $Pr_w = \{x_{s_c} \cdot x_{w_l} + f(x_{w_l}) \cdot x_{s_c}|_{e_w} \rightarrow 1|x_{w_l},$
  $x_{s_c} \cdot x_{w_r} + f(x_{w_r}) \cdot x_{s_c}|_{e_w} \rightarrow 1|x_{w_r}\};$
  $Pr_{s_c} = \{x_{s_c} \rightarrow 1|x_{s_c}\};$
  $Pr_{c_i} = \{x_{c_i,s_l} \cdot x_{c_i,w_l}\Big|_{e_{c_i}} \rightarrow 1\Big|x_{s_l},$
  $x_{c_i,s_r} \cdot x_{c_i,w_r}\Big|_{e_{c_i}} \rightarrow 1\Big|x_{s_r}\}, 1 \leqslant i \leqslant k;$
  $Pr_{s_i} = \{3x_{s_i,i} \rightarrow 1|x_{s_i,i} + 1|x_{c_i,s_l} + 1|x_{c_i,s_r}\};$
  $1 \leqslant i \leqslant k;\ Pr_{w_i} = \{2x_{w_i,w_l}\Big|_{e_{w_i}} \rightarrow 1\Big|x_{w_i,w_l}$
  $+ 1|x_{c_i,w_l},\ 2x_{w_i,w_r}\Big|_{e_{w_i}} \rightarrow 1\Big|x_{w_i,w_r} + 1|x_{c_i,w_r}\},$
  $1 \leqslant i \leqslant k$

The meaning of the variables from the model is the following:

- x_{s_l} and x_{s_r} from the region s represent *leftSpeed* and *rightSpeed*, the sum of the products are accumulated in s after applying the rules from the membranes Pr_{c_i}, where $1 \leqslant i \leqslant k$;
- x_{s_c} from the compartment s_c is *cruiseSpeed*;
- each pair of weights, $weightLeft_i$ and $weightRight_i$, resides in the regions w_i, $1 \leqslant i \leqslant k$;
- for each proximity sensor, $prox_i$, a compartment is defined, namely s_i, containing a single variable, $x_{s_i,i}$, $1 \leqslant i \leqslant k$;
- the products are calculated by two distinct programs, $weightLeft_i \cdot prox_i$, and $weightRight_i \cdot prox_i$, $1 \leqslant i \leqslant k$, in the compartments c_i.

As presented in EN P system definition, all the programs that satisfy the enzymatic condition are applied in each simulation step. Moreover, according to Remark 3, we opted to reset the values of the speeds x_{s_l} and x_{s_r} by adding them multiplied by zero in the production function from the region s. In this way, the previous values of *leftSpeed* and *rightSpeed* are consumed at each step. This approach will also be used in the models presented later in this paper.

4.2. Refined model

During the test phase of the above presented model, we observed that even though the robot avoids the obstacles (road borders, in case of generating roads with AmbieGen), it tends to have a "zig-zag" motion going from the proximity of a border to the proximity of the other one. Considering this, an immediate adjustment was to recenter the robot after avoiding an obstacle and reaching the center of the road, so the robot will go straight until it encounters a new obstacle.

We made this adjustment by introducing a new membrane, called *Direction*. The membrane has seven variables, called *directionLeft*, *directionRight*, *angle*, *state*, *distance*, *angleStep*, and *distanceStep*, which will be detailed when giving the formal definition of the model. In order to obtain the desired behavior, we used differential drive kinematics equations.[54] The *state* variable aims to reproduce a finite state machine inside the production function of the membrane, with the following states:

1. state 0—the robot is moving in a straight line
2. state 1—the robot is moving in the presence of an obstacle
3. state 2—the robot is moving to approximately the center of the lane
4. state 3—the robot is recentering on the lane. Before introducing the mathematical definition of the described model, we firstly defined four functions needed in the production functions, as stated in Table 1.

Also, we used the constant *len* which represents the axle length of the robot. For E-puck this is equal to 52 mm.

The model is defined as follows:

$$\Pi_{M_3} = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0))) \tag{15}$$

where

- $m = 3k + 4, k = 6;$
- $H = \{s, d, w, s_c\} \cup \bigcup_{(i=1)}^{k} c_i, s_i, w_i;$
- $\mu = [[[[[]_{s_1} []_{w_1}]_{c_1} \ldots [[]_{s_k} []_{w_k}]_{c_k} []_{s_c}]_w]_d]_s;$
- $Var_s = \{x_{s_l}, x_{s_r}\}, Var_d = \{x_{d_l}, x_{d_r}, x_a, x_{st}, x_{dst}, x_{as}, x_{ds}, e_{ds}, e_{dw}, e_d\}, Var_w = \{x_{w_l}, x_{w_r}, e_w\}, Var_{s_c} = \{x_{s_c}\}, Var_w = \{x_{w_l}, x_{w_r}, e_w\}, Var_{s_c} = \{x_{s_c}\},$
  $Var_{c_i} = \{x_{c_i,s_l}, x_{c_i,s_r}, x_{c_i,w_l}, x_{c_i,w_r}, e_{c_i}\}, 1 \leqslant i \leqslant k,$
  $Var_{s_i} = \{x_{s_i,i}\}, 1 \leqslant i \leqslant k,$
  $Var_{w_i} = \{x_{w_i,w_l}, x_{w_i,w_r}, e_{w_i}\}, 1 \leqslant i \leqslant k;$
- $Var_i(0) = 0, i \in \{s, w, s_c\} \cup \bigcup_{i=1}^{k} \{c_i, s_i, w_i\},$
  $Var_{d_{e_d}}(0) = 100, Var_{d_i}(0) = 0, i \in Var_d \setminus \{e_d\};,$
  $Var_{d_{e_d}}(0) = 100, Var_{d_i}(0) = 0, i \in Var_d \setminus \{e_d\};$
- $Pr_s = \{0 \cdot x_{s_l} \cdot x_{s_r} \rightarrow 1|x_{s_l} + 1|x_{s_r}\};$
- $Pr_d = \{not(x_{st}, 3) \cdot x_{d_l} + eq(x_{st}, 3) \cdot ((x_{d_l} + |x_{as}|) \cdot sgn(x_{as}))|_{e_{ds}} \rightarrow 1|x_{s_l},$
  $not(x_{st}, 3) \cdot x_{d_r} + eq(x_{st}, 3) \cdot ((x_{d_r} + |x_{as}|) \cdot sgn(-x_{as}))|_{e_{ds}} \rightarrow 1|x_{s_r},$
  $gt(x_{as}, 0) \cdot x_{as}|_{e_{ds}} \rightarrow 1|x_{as},$
  $x_{d_l}|_{e_{dw}} \rightarrow 1|x_{s_l},$
  $x_{d_r}|_{e_{dw}} \rightarrow 1|x_{s_r},$
  $e_{ds} \cdot e_{dw} \cdot 0|_{e_d} \rightarrow 1|e_{ds} + 1|ed_w,$
  $not(x_{d_r} - x_{d_l}, 0) \cdot \frac{x_{d_r} - x_{d_l}}{2len} + eq(x_{d_r} - x_{d_l}, 0) \cdot x_a + x_a \cdot 0|_{e_{dw}} \rightarrow 1|x_a,$
  $not(x_{d_r} - x_{d_l}, 0) \cdot \frac{x_{d_r} - x_{d_l}}{4} + eq(x_{d_r} - x_{d_l}, 0) \cdot x_{as}|_{e_{dw}} \rightarrow 1|x_{as}, \cdot x_{as}|_{e_{dw}} \rightarrow 1|x_{as},$
  $not(x_{d_r} - x_{d_l}, 0) \cdot \lfloor |\frac{4}{2len}| \rfloor \cdot 3 + eq(x_{d_r} - x_{d_l}, 0) \cdot x_{ds} + x_{ds} \cdot 0|_{e_{dw}} \rightarrow 1|x_{ds},$
  $eq(x_{st}, 1) \cdot 2 + eq(x_{st}, 2) \cdot not(x_{dst}, 0) \cdot 2 + eq(x_{st}, 2) \cdot eq(x_{dst}, 0) \cdot 3 + eq(x_{st}, 3) \cdot gt(x_{dst}, 0) \cdot 3 + eq(x_{st}, 3) \cdot eq(x_{dst}, 0) \cdot 0|_{e_{ds}} \rightarrow 1|x_{st}, \quad x_{st} \cdot 0 + 1|_{e_{dw}} \rightarrow 1|x_{st},$
  $x_{dst} \cdot 0 + 180|_{e_{dw}} \rightarrow 1|x_{dst},$
  $eq(x_{st}, 2) \cdot gt(x_{dst}, 0) \cdot (x_{dst} - 1) + eq(x_{st}, 1) \cdot x_{dst}|_{e_{ds}} \rightarrow 1|x_{dst},$
  $eq(x_{st}, 3) \cdot gt(x_{ds}, 0) \cdot (x_{ds} - 1) + not(x_{st}, 3) \cdot x_{ds}|_{e_{ds}} \rightarrow 1|x_{ds};$
  $Pr_w = \{x_{s_c} \cdot x_{w_l}|_{e_w} \rightarrow 1|x_{d_l},$
  $f(x_{w_l}) \cdot x_{s_c}|_{e_w} \rightarrow 1|x_{d_l}\},$
  $x_{s_c} \cdot x_{w_r}|_{e_w} \rightarrow 1|x_{d_r},$
  $f(x_{w_r}) \cdot x_{s_c}|_{e_w} \rightarrow 1|x_{d_r}\}$
  $(f(x_{w_l}) \cdot f(x_{w_r})) \cdot x_{s_c} \cdot 100|_{e_w} \rightarrow 1|e_{ds},$
  $(not(x_{w_l}, 0) \cdot not(x_{w_r}, 0)) \cdot x_{s_c} \cdot 100|_{e_w} \rightarrow 1|e_{dw};$
  $Pr_{s_c} = \{x_{s_c} \rightarrow 1|x_{s_c}\};$

$$\begin{aligned} Pr_{c_i} &= \{x_{c_i,s_l} \cdot x_{c_i,w_l}\Big|_{e_{c_i}} \rightarrow 1\Big|x_{s_l}, \\ &\quad x_{c_i,s_r} \cdot x_{c_i,w_r}\Big|_{e_{c_i}} \rightarrow 1\Big|x_{s_r}\}, 1 \leqslant i \leqslant k; \\ Pr_{s_i} &= \{3x_{s_i,i} \rightarrow 1|x_{s_i,i} + 1|x_{c_i,s_l} + 1|x_{c_i,s_r}\}, 1 \leqslant i \leqslant k; \\ Pr_{w_i} &= \{2x_{w_i,w_l}\Big|_{e_{w_i}} \rightarrow 1\Big|x_{w_i,w_l} + 1|x_{c_i,w_l}, \\ &\quad 2x_{w_i,w_r}\Big|_{e_{w_i}} \rightarrow 1\Big|x_{w_i,w_r} + 1|x_{c_i,w_r}\}, 1 \leqslant i \leqslant k; \end{aligned}$$

As observed, the main difference in the membrane's structure is made by the new region called *direction*. Next we present the meaning of the variables used in it:

- x_{d_l} and x_{d_r} represent *directionLeft* and *direction Right*;
- x_a and x_{as} represent *angle* and *angleStep*;
- x_{dst} and x_{ds} represent *distance* and *distanceStep*;
- x_{st} represents the *state* of the simulated finite state machine

4.3. Extended refined model

Experiments carried out with the second model proved that when the robot approaches perpendicularly the obstacle, it remains locked into that obstacle. This limitation is caused by the values of weights, which have values of opposite sign. This situation can be easily explained by the position of the proximity sensors that are facing the obstacle perpendicularly.

To solve this problematic behavior we introduced Π_{M_4}, a model with the same membrane structure as Π_{M_3}, but containing two more production functions inside *Weight* compartment. These functions are distributed to *directionRight* and *directionLeft* variables, as follows:

$$directionLeft = eq(|weightLeft|, |weightRight|) \cdot gt(|weightLeft|, 0) \cdot gt(|weightRight|, 0) \cdot weightLeft \cdot cruiseSpeed \cdot 0$$

$$directionRight = eq(|weightLeft|, |weightRight|) \cdot gt(|weightLeft|, 0) \cdot gt(|weightRight|, 0) \cdot weightRight \cdot 0 + cruiseSpeed$$

In this way, we ensure that the robot will not get stuck when perpendicularly facing an obstacle, as the speed of the right wheel (guided by *directionRight* variable) will move the robot to the left.

An interesting step would be to automatically decide what direction to follow in this situation (e.g., if an obstacle is near on the left, a better decision should be to activate the *directionLeft* variable, thus moving the robot to the right).

4.4. Linear regression-based model

One of the most striking limitations of all the previous models was the empirical nature of weights. Choosing the values of these parameters in a "trial and error" manner, taking into consideration that values of opposite sign should be assigned to each sensor value, can lead to convincing results, as we can observe looking at the previous models. Nevertheless, the choice of these parameters can be made using linear regression.

Table 1. Functions used in the model.

sgn(x)	$sgn(x) = \begin{cases} 1, & if\ x = 1 \\ 0, & if\ x = 0 \\ -1, & if\ x = -1 \end{cases}$
not(x, y)	$not(x,y) = \begin{cases} 1, if\ x \neq y \\ 0, otherwise \end{cases}$
gt(x, y)	$gt(x,y) = \begin{cases} 1, if\ x > y \\ 0, otherwise \end{cases}$
eq(x, y)	$eq(x,y) = \begin{cases} 1, if\ x = y \\ 0, otherwise \end{cases}$

Taking into consideration the values returned by applying linear regression to the data retrieved using the obstacle avoidance Webots controller, we obtained a model with a simplified structure, implementing the linear relationship between a dependent value and six independent values. Before introducing the formal definition of the model, let us present this relationship:

$$leftSpeed = a_0 + \sum_{i=1}^{n} a_i \cdot prox_i \tag{16}$$

$$rightSpeed = b_0 + \sum_{i=1}^{n} b_i \cdot prox_i \tag{17}$$

The above equations, two first-degree polynomials, describe the relationship between the instantaneous speed of the wheels and the values retrieved from the proximity sensors. Analyzing the linear regression model, in each equation we have one dependent value ($leftSpeed, rightSpeed$) and n independent values, representing the values retrieved from each proximity sensor, $prox_i$.

As mentioned earlier in this section, this model has a simpler structure which will be discussed next, after introducing the model as follows:

$$\begin{aligned} \Pi_{M_5} = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, \\ (Var_m, Pr_m, Var_m(0))) \end{aligned} \tag{18}$$

where

- $m = k + 4, k = 6;$
- $H = \{s, p_l, p_r, c\} \cup \bigcup_{i=1}^{k} \{s_i\};$

Table 2. Experimental results.

Test type	Π_{M_1}	Π_{M_2}	Π_{M_3}	Π_{M_4}	Π_{M_5}
Arena with obstacles	Failed	Passed	Passed	Passed	Passed
Corridor straight	Failed	Failed	Failed	Passed	Passed
Corridor angle	Failed	Passed	Failed	Passed	Passed
Square straight	Failed	Failed	Passed	Passed	Passed
Square angle	Failed	Passed	Passed	Passed	Passed
Road 1	Failed	Passed	Passed	Failed	Passed
Road 2	Failed	Passed	Passed	Passed	Passed
Road 3	Failed	Passed	Passed	Failed	Passed
Road 4	Passed	Passed	Passed	Failed	Passed

- $\mu = [[[]_{s_1} \cdots []_{s_k}, []_{p_l}, []_{p_r}]_c]_s$;
- $Var_s = \{x_{s_l}, x_{s_r}\} Var_{s_i} = \{x_{s_r,i}\}, 1 \leqslant i \leqslant k,$
  $Var_{p_r} = \{a_i, e_a\}, 0 \leqslant i \leqslant k,$
  $Var_{p_l} = \{b_i, e_b\}, 0 \leqslant i \leqslant k,$
  $Var_c = \{x_{s_v,i}, e_c\}, 1 \leqslant i \leqslant k;$
- $Var_i(0) = 0, i \in \{s, p_l, p_r, c\} \cup \bigcup_{i=1}^{k} \{s_i\}$;
- $Pr_s = \{0 \cdot x_{s_l} \cdot x_{s_r} \rightarrow 1|x_{s_l} + 1|x_{s_r}\}$;
- $Pr_{s_i} = \{2 \cdot x_{s_r,i} \rightarrow 1|x_{s_r,i} + 1|x_{s_v,i}\}$;
- $Pr_{p_r} = \{a_i|_{e_a} \rightarrow 1|a_i\}, 0 \leqslant i \leqslant k$;
- $Pr_{p_l} = \{b_i|_{e_b} \rightarrow 1|b_i\}, 0 \leqslant i \leqslant k$;
- $Pr_c = \{a_0 + \sum_{i=1}^{k} a_i$
  $\cdot x_{s_v,i}|_{e_c} \rightarrow 1|x_{s_r}, b_0 + \sum_{i=1}^{k} b_i \cdot x_{s_v,i}|_{e_c} \rightarrow 1|x_{s_l}\}$

The meaning of the variables from the model is the following:

- x_{s_l} and x_{s_r} from the region s represent *leftSpeed* and *rightSpeed*;
- for each proximity sensor, $prox_i$, a compartment is defined, namely s_i, containing a single variable, $x_{s_r,i}$, $1 \leqslant i \leqslant k$;
- the parameters values, a_i, b_i, $0 \leqslant i \leqslant k$, are then persisted inside p_r and p_l compartments;
- $x_{s_v,i}$ represents the actual sensor values and is multiplied with the parameter values inside c compartment.

Π_{M_5} describes the membrane structure of the model, consisting of 10 membranes, one for each sensor (we used six proximity sensors placed radially at the frontal part of the robot), one for the speed, two for the parameters of the left and right sensors values, and one for computing the speed of each wheel. We used enzymes inside the parameter membranes in order to execute each rule which persists the values of the parameters. These parameters are then used to simultaneously execute (also in the presence of an enzyme) the formulas presented in 10 and 11 inside the compute region.

5. Simulation results

This section illustrates the simulation and testing phase, presenting scenarios designed to challenge the robot as well as their results. Considering the way we defined and integrated the models, these simulation scenarios guided us to refine intermediate variants of the model based on the results.

In this experiment we opted for four scenarios and all of them are defined by the type of the area E-puck needs to go through. These are introduced as follows:

- *a circular arena with seven obstacles*
- *corridors*
- *a square*
- *roads generated by AmbieGen*

The first three scenarios were created in a simple manner, using the areas that E-puck should cover, being defined using Webots-embedded shapes. For the first one we used seven boxes as obstacles, having uniform dimensions of 20 × 20 × 20 cm. For the last scenario, we used AmbieGen and then selected roads with different aspects. Some generation parameters, such as map size or generation time can be easily set from the command line, when using the tool. Other types of parameters (e.g., minimum distance of a generated road, maximum distance to go straight during a road) are constant values in the source code and are easily configurable. More details about AmbieGen were presented in the study by Bobe et al.[29] and Gambi et al.[55]

As mentioned before, this approach encapsulates four main types of scenarios, *a circular arena with obstacles*, *a corridor*, *a square*, and *roads generated by AmbieGen*. For *corridor* and *square* we simulated two situations. In the first situation, the robot starts in a straight line from the middle of the corridor or from the middle of the square. For simplicity, let us call this types of tests *corridor straight* and *square straight*. The other situation assumes that the robot also starts from the middle of the corridor, respectively square, but with an angle of 15°. Let us refer to these test types as *corridor angle* and *square angle*.

Table 2 contains the simulation results for each model instance, starting with the basic one, Π_{M_1}. This table was taken from the initial conference article[27] and extended with the results for Π_{M_5}. For a better understanding, we

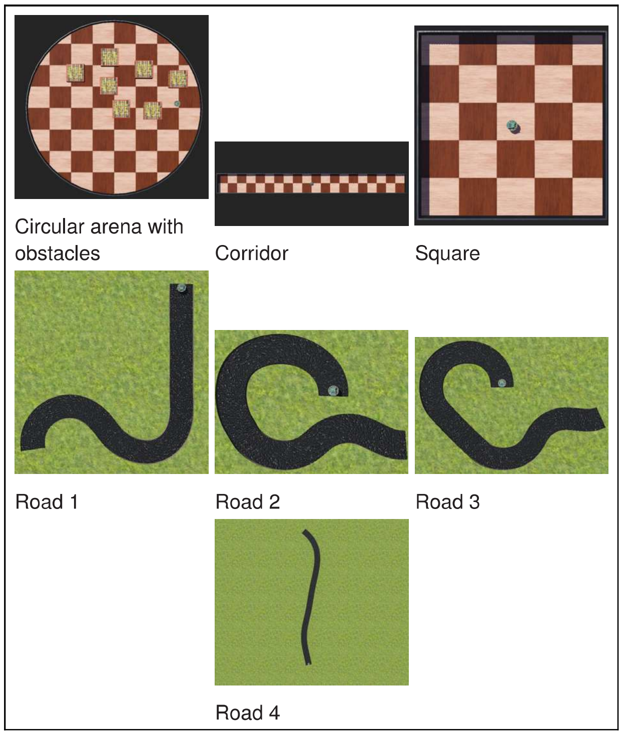

Figure 2. Test cases as represented in Webots.

consider necessary to include it in the actual version. It presents the test types presented in the above paragraph and four roads generated by AmbieGen and imported to Webots. We chose these roads from a larger suite, opting for different curvatures in order to better observe the behavior of each model.

When analyzing the experimental results, we noticed that the basic model, Π_{M_1}, has the worst performance compared to the other four models. This basic model passed just one test, Road 4, which is the simplest one (as presented in Figure 2).

Another observation that can be extracted from the experimental results recorded in Table 2 is that the second model, Π_{M_2}, has an improved performance in terms of passing the road tests. The improvement consists in a rotation that is performed when the road is curved (i.e., the proximity sensors detect an obstacle). Additional road tests were added to.[56]

The refocusing movement added to the behavior of the third model, Π_{M_3}, conducted to a natural movement on the roads, all the roads being also passed for this model. Nevertheless, it can be observed that this modification of the membrane structure (with the addition of the membrane *direction*) made Π_{M_3} to fail the *corridor angle* challenge, but pass the *square straight*, failed by Π_{M_2}. Π_{M_4} came with an improvement in passing the corridor and square tests, due to the property of moving even if the obstacle is in the front of the robot (placed perpendicularly), the situation when the robot gets stuck in front of

Table 3. Models performance on road tests.

Test type	Π_{M_1}	Π_{M_2}	Π_{M_3}	Π_{M_4}	Π_{M_5}
Set 1	0	5	5	0	5
Set 2	0	5	5	2	5
Set 3	0	5	5	0	5
Set 4	0	5	5	0	5
Set 5	0	5	5	3	5
Total	0	25	25	5	25

an obstacle being handled. However, the adjustments made came with a few disadvantages in passing road tests, the robot moving to the left and failing the test at the moment the robot direction is perpendicular to one of the borders. In many instances, this scenario was frequently observed, serving as the underlying cause for the model's failure to pass three out of the four road test examples.

By analyzing the results produced by Π_{M_5}, we can affirm that even if this test cases were designed to trigger possible limitations of the model involved, the linear regression-based model passed all the tests. Figure 2 illustrates the experimental tests discussed above. A video representation of each model performance on these tests can be found in the URL provided.[57]

As stated in the "Methodology Description and Research Environment" section, we can divide our experiments into two different types, based on their purpose. The above presented were designed to highlight the limitations of each model instance. Next, we introduce the second stage of simulations, aimed to demonstrate the validity of each model version.

Considering a higher number of tests, Table 3 illustrates the results obtained after running five sets of tests generated by varying parameters of road generation in AmbieGen. Thus, starting from small values, each set used incremented values of the following three parameters: minimum distance of a generated road, maximum distance to go straight during a road, and the map size. The values transmitted to AmbieGen for these parameters are expressed in meters. After running the tool with different values for this three parameters, we obtained 25 roads with various characteristics, 5 for each set. The parameters defining each set are the following:

- $Set1 : (min\_len : 5, max\_len : 15, map\_size : 150)$
- $Set2 : (min\_len : 10, max\_len : 30, map\_size : 200)$
- $Set3 : (min\_len : 15, max\_len : 40, map\_size : 400)$
- $Set4 : (min\_len : 20, max\_len : 60, map\_size : 500)$
- $Set5 : (min\_len : 100,$ $max\_len : 150, \ map\_size : 1000)$

Highlighting the observations derived from the data presented in the above table, we can conclude that Π_{M_1} has

Table 4. Performance metrics of the models.

Model	*VarN*	*ProgN*	*Road2T*	*Double–Road2T.*	*Increase* (%)
Π_{M_2}	47	33	120	232	93
Π_{M_3}	54	52	130	242	86
Π_{M_4}	54	54	136	238	75
Π_{M_5}	28	23	159	298	87

the worst performance, being unable to pass roads with curves. Even if the model was able to traverse a considerable part of some road tests, the model failed to pass the curves. As expected, Π_{M_2} passed all the tests, the reason that led us to design Π_{M_3} being the "zig-zag" motion of the robot during the tests which tends to seem unnatural. Next, we can observe that Π_{M_3} also passed all the tests, but its extension, Π_{M_4}, encountered real problems. This can be explained by its static nature as when perpendicularly approaching a road border and then moving to the left, often ending up blocked in the margins.

Our last model, Π_{M_5}, came as a better model, apart from passing all the tests, having a visibly simpler structure than the previous ones. Also, the empirical component has been excluded, the parameter values being determined using an elaborate heuristic.

5.1. Performance metrics

We have made some more experiments with respect to some performance metrics. We have considered one of the tests previously discussed, Road 2 in Figure 2. Only models $\Pi_{M_i}, 2 \leqslant i \leqslant 5$, are considered, as the first model fails to run the test and this is just the starting model utilized to develop the others. For each of these four models, the following performance metrics are defined: number of variables ($VarN$), number of programs ($ProgN$), the execution time for running the Road 2 test as defined above ($Road2T$), and for the case of doubling its length ($Double - Road2T$). We have also expressed in percentages the increase in time when the road length doubles. The results of these metrics are presented in Table 4. The time is expressed in seconds.

One can notice the two metrics expressing the static values of memory used by the four models, defined by the number of variables and programs, $VarN$ and $ProgN$, respectively, are consistent with the models' structure as presented previously. Model Π_{M_2} is just a simple extension of model Π_{M_1}. Models Π_{M_3} and Π_{M_4} present a significant change of the $VarN$ and $ProgN$, compared to the same values for Π_{M_2}. These changes are explained by the additional functionality introduced for handling the movement and direction. Finally, model Π_{M_5} shows a minimum of these values, explained by the conceptual changes of the model presented above in this paper. The metrics related to the execution time, both $Road2T$ and $Double - Road2T$, show that going from one model i to the next one, $i + 1$, some overhead is introduced and even when static structural changes are made the time is not reduced. Also, when the length of the road is increased, their execution time increases similarly.

6. Conclusions and future work

A methodology based on a test-driven approach has been introduced in this paper. The methodology entails the creation of new model instances based on relevant simulations that highlight the behavior of each version of the model. As we already introduced four variants of model in the conference paper,[27] this article presented the working heuristic, as well as a new performant instance of the model. The testing stage has also been refined, generating more road tests with different characteristics by varying the parameters of AmbieGen, the search-based software testing tool. As stated above, the last version of the model, which was introduced in this paper, exhibits a great number of advantages, including the performance during the test stage, but also the simple structure and the non-empirical manner of setting the weights.

In terms of further development, we note the possibility of exploring other types of P systems applied in robotics. Another objective is to dynamically assign weight values during the road test, guiding the robot to take a decision when it is close to an obstacle and another is nearby. Furthermore, we will explore the potential for overcoming the limitations imposed by existing tools for simulating other types of P systems.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

ORCID iD

Radu Traian Bobe https://orcid.org/0009-0005-6611-3176

References

1. Păun G. *Membrane computing: an introduction*. Cham: Springer Science & Business Media, 2002.

2. Păun G, Rozenberg G and Salomaa A (eds). *The Oxford handbook of membrane computing*. Oxford: Oxford University Press, 2010.
3. Zhang G, Pérez-Jiménez MJ and Gheorghe M. *Real-life applications with membrane computing*. Cham: Springer, 2017.
4. Yu W, Wu J, Chen Y, et al. Fuzzy tissue-like P systems with promoters and their application in power coordinated control of microgrid. *J Membr Comput* 2023; 5: 1–11.
5. Valencia-Cabrera L and Song B. Tissue P systems with promoter simulation with MeCoSim and P-lingua framework. *J Membr Comput* 2020; 2: 95–107.
6. de la Cruz RTA, Cabarle FGC, Macababayao ICH, et al. Homogeneous spiking neural P systems with structural plasticity. *J Membr Comput* 2021; 3: 10–21.
7. Gheorghe M, Lefticaru R, Konur S, et al. Spiking neural P systems: matrix representation and formal verification. *J Membr Comput* 2021; 3: 133–148.
8. Qiu C, Xue J, Liu X, et al. Deep dynamic spiking neural P systems with applications in organ segmentation. *J Membr Comput* 2022; 4: 329–340.
9. Verlan S, Freund R, Alhazov A, et al. A formal framework for spiking neural P systems. *J Membr Comput* 2020; 2: 355–368.
10. Yu W, Xiao X, Wu J, et al. Application of fuzzy spiking neural dP systems in energy coordinated control of multi-microgrid. *J Membr Comput* 2023; 5: 69–80.
11. Zhao S, Zhang L, Liu Z, et al. ConvSNP: a deep learning model embedded with SNP-like neurons. *J Membr Comput* 2022; 4: 87–95.
12. Orellana-Martín D, Valencia-Cabrera L, Riscos-Núñez A, et al. P systems with proteins: a new frontier when membrane division disappears. *J Membr Comput* 2019; 1: 29–39.
13. Sosík P. P systems attacking hard problems beyond NP: a survey. *J Membr Comput* 2019; 1: 198–208.
14. Păun G and Păun R. Membrane computing and economics: numerical P systems. *Fundam Inform* 2006; 73: 213–227, http://content.iospress.com/articles/fundamenta-informaticae/fi73-1-2-20
15. Pavel A, Arsene O and Buiu C. Enzymatic numerical P systems—a new class of membrane computing systems. In: *2010 IEEE fifth international conference on bio-inspired computing: theories and applications (BIC-TA)*, Changsha, China, 23–26 September 2010, pp. 1331–1336. New York: IEEE. DOI: 10.1109/BICTA.2010.5645071.
16. Wang X, Zhang G, Neri F, et al. Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integr Comput-Aided Eng* 2016; 23: 15–30.
17. Pérez-Hurtado I, Martínez-del Amor MA, Zhang G, et al. A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. *Integr Comput-Aided Eng* 2020; 27: 121–138.
18. Buiu C and Florea AG. Membrane computing models and robot controller design, current results and challenges. *J Membr Comput* 2019; 1: 262–269.
19. Florea AG and Buiu C. PeP—an open-source software simulator of numerical P systems and numerical P systems with enzymes, 2017, https://github.com/andrei91ro/pep
20. Florea AG and Buiu C. Modelling multi-robot interactions using a generic controller based on numerical P systems and ROS. In: *2017 9th international conference on electronics, computers and artificial intelligence (ECAI)*, Targoviste, 29 June–1 July 2017, pp. 1–6. New York: IEEE.
21. Khari M and Kumar P. An extensive evaluation of search-based software testing: a review. *Soft Comput* 2019; 23: 1933–1946.
22. Ţurlea A, Gheorghe M, Ipate F, et al. Search-based testing in membrane computing. *J Membr Comput* 2019; 1: 241–250.
23. Humeniuk D, Antoniol G and Khomh F. AmbieGen tool at the SBST 2022 tool competition. In: *Proceedings of the 15th workshop on search-based software testing*, Pittsburgh, PA, 9 May 2022, pp. 43–46. New York: Association for Computing Machinery.
24. George B and Williams L. A structured experiment of test-driven development. *Inf Softw Technol* 2004; 46: 337–342.
25. Mondada F, Bonani M, Raemy X, et al. The e-puck, a robot designed for education in engineering. In: *Proceedings of the 9th conference on autonomous robot systems and competitions*, Castelo Branco, 7 May 2009, pp. 59–65. Castelo Branco: Instituto Politécnico de Castelo Branco (IPCB).
26. Michel O. Cyberbotics Ltd. Webots™: professional mobile robot simulation. *Int J Adv Robot Syst* 2004; 1: 5.
27. Bobe RT, Gheorghe M, Ipate F, et al. Test-driven simulation of robots controlled by enzymatic numerical P systems models. In: Guisado-Lizar JL, Riscos-Núñez A, Morón-Fernández MJ, et al. (eds) Simulation tools and techniques. Cham: Springer Nature, pp. 56–69.
28. EAI SIMUtools 2023, https://simutools.eai-conferences.org/2023/
29. Bobe RT, Ipate F and Niculescu IM. Modelling and search-based testing of robot controllers using enzymatic numerical P systems. In: Cheval H, Leuştean L and Sipoş A (eds) *Proceedings 7th symposium on working formal methods*, Bucharest, Romania, 21-22 September 2023, Electronic proceedings in theoretical computer science, vol 389. Waterloo, NSW, Australia: Open Publishing Association, pp. 1–10.
30. Kavitha S, Varuna S and Ramya R. A comparative analysis on linear regression and support vector regression. In: *2016 online international conference on green engineering and technologies (IC-GET)*, Coimbatore, India, 19 November 2016, pp. 1–5. New York: IEEE.
31. Groß J. *Linear regression*, vol 175. Cham: Springer Science & Business Media, 2003.
32. Maulud D and Abdulazeez AM. A review on linear regression comprehensive in machine learning. *J Appl Sci Technol Trends* 2020; 1: 140–147.
33. Stöter FR, Chakrabarty S, Edler B, et al. Classification vs. regression in supervised learning for single channel speaker count estimation. In: *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, Calgary, AB, Canada, 15–20 April 2018, pp. 436–440. New York: IEEE.
34. Dielman TE. *Applied regression analysis for business and economics*. Pacific Grove, CA: Duxbury/Thomson Learning, 2001.

35. Chávez-Olivares CA, Reyes-Cortes F, González-Galván EJ, et al. Experimental evaluation of parameter identification schemes on a direct-drive robot. *Proc Inst Mech Eng Part I J Syst Control Eng* 2012; 226: 1419–1431.
36. Sigaud O, Salaün C and Padois V. On-line regression algorithms for learning mechanical models of robots: a survey. *Robot Auton Syst* 2011; 59: 1115–1129.
37. Nguyen-Tuong D and Peters J. Model learning for robot control: a survey. *Cognit Process* 2011; 12: 319–340.
38. Bona B and Curatella A. Identification of industrial robot parameters for advanced model-based controllers design. In: *Proceedings of the 2005 IEEE international conference on robotics and automation*, Barcelona, 18–22 April 2005. New York: IEEE.
39. Lizana J, Perejón A, Sanchez-Jimenez PE, et al. Advanced parametrisation of phase change materials through kinetic approach. *J Energy Storage* 2021; 44: 103441.
40. Quan X, Luo D, Yang Q, et al. Multidimensional graph transformer networks for trajectory prediction in urban road intersections. *J Membr Comput*. Epub ahead of print 9 July 2024. DOI: 10.1007/s41965-024-00161-0.
41. Aerts A, Reniers M and Mousavi MR. Model-based testing of cyber-physical systems. In: Song H, Rawat DB, Jeschke S, et al. (eds) *Cyber-physical systems*. Amsterdam: Elsevier, 2017, pp. 287–304.
42. Turlea A. Search based model in the loop testing for cyber physical systems. In: *2018 IEEE 16th international conference on embedded and ubiquitous computing (EUC)*, Bucharest, 29–31 October 2018, pp. 22–28. New York: IEEE.
43. McMinn P. Search-based software testing: past, present and future. In: *2011 IEEE fourth international conference on software testing, verification and validation workshops*, Berlin, 21–25 March 2011, pp. 153–163. New York: IEEE.
44. Deb K, Pratap A, Agarwal S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans Evolut Comput* 2002; 6: 182–197.
45. Arrieta A, Wang S, Markiegi U, et al. Search-based test case generation for cyber-physical systems. In: *2017 IEEE congress on evolutionary computation (CEC)*, Donostia, 5–8 June 2017, pp. 688–697. New York: IEEE.
46. Togelius J, Champandard AJ, Lanzi PL, et al. Procedural content generation: goals, challenges and actionable steps. *Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik*. DOI:10.4230/DFU.Vol6.12191.61
47. Gambi A, Mueller M and Fraser G. Automatically testing self-driving cars with search-based procedural content generation. In: *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, Beijing, China, 15–19 July 2019, pp. 318–328. New York: ACM.
48. Arnold J and Alexander R. Testing autonomous robot control software using procedural content generation. In: *Computer safety, reliability, and security: 32nd international conference, SAFECOMP 2013*, Toulouse, 24–27 September 2013, pp. 33–44, proceedings 32. Cham: Springer.
49. Webots reference manual, https://cyberbotics.com/doc/reference/proto
50. Humeniuk D, Khomh F and Antoniol G. AmbieGen: a search-based framework for autonomous systems testing. *arXiv preprint arXiv*:230101234, 2023. DOI:10.48550/arXiv.2301.01234
51. Whitley D, Rana S, Dzubera J, et al. Evaluating evolutionary algorithms. *Artif Intell* 1996; 85: 245–276.
52. Webots obstacle avoidance controller, https://cyberbotics.-com/doc/guide/tutorial-4-more-about-controllers
53. Github project, https://github.com/radubobe/Research/tree/main/Modelling%20and%20testing%20robot%20controllers%20using%20ENPS
54. Hellstrom T. *Kinematics equations for differential drive and articulated steering*. Umea University, 2011, https://www8.cs.umu.se/kurser/5DV122/HT13/material/Hellstrom-ForwardKinematics.pdf
55. Gambi A, Jahangirova G, Riccio V, et al. SBST tool competition 2022. In: *Proceedings of the 15th workshop on search-based software testing*, Pittsburgh, PA, 9 May 2022, pp. 25–32. New York: IEEE.
56. Github simulation results folder, https://github.com/radubobe/Research/tree/main/Modelling%20and%20testing%20robot%20controllers%20-using%20ENPS/Simulation%20results
57. Simulation of E-puck controlled by enzymatic numerical P systems models in Webots, https://youtu.be/FA7snrqaKKs

Author biographies

Radu Traian Bobe is a PhD student and University Assistant in the Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest.

Marian Gheorghe is a Professor Emeritus of Computational Models and Software Engineering at University of Bradford. He was Head of the Department of Computer Science (2018–2020).

Florentin Ipate is a Full Professor in the Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest. He is an editor of Springer's Journal of Membrane Computing and also member of the steering committee of Conference on Membrane Computing and Bulletin committee of the International Membrane Computing Society.

Ionuţ Mihai Niculescu is a researcher and computer scientist. His PhD thesis was distinguished with The PhD Thesis of the Year 2018 by the International Membrane Computing Society (IMCS).