Probabilistic Verification of Robot Controllers using Enzymatic Numerical P Systems Models

Radu Traian Bobe[1][0009−0005−6611−3176], Florentin Ipate[1][0000−0001−8777−3425], and Savas Konur[2][0000−0003−2704−8381]

[1] Department of Computer Science, Faculty of Mathematics and Computer Science University of Bucharest, Str Academiei 14, Bucharest, 010014, Romania
radu.bobe@s.unibuc.ro; florentin.ipate@unibuc.ro
[2] School of Computer Science and Digital Technologies, Aston University, Birmingham, B4 7ET, United Kingdom
s.konur@aston.ac.uk

Abstract. The validation of robot controllers constitutes a complex problem: traditional testing and simulation examine only specific execution traces, providing practical evidence without formal guarantees, while exhaustive formal verification becomes computationally intractable due to state space explosion problem. This paper addresses this challenge by presenting a practical approach for verification of obstacle avoidance robot controllers that are based on enzymatic numerical P systems (ENPS). The study demonstrates how model checking can be effectively applied to examine and evaluate the probabilistic behaviour of the controllers. Several properties are formally verified with Monte Carlo Model Checker (MC2) using a logic language called Probabilistic Linear-time Temporal Logic (PLTL), which incorporates temporal operators for describing how the system evolves over time. The results proved that the ENPS model used in our experiments confers the main capabilities to the controllers. Furthermore, we validated that the dynamic generation of weight parameters that are assigned to each proximity sensor of the robot does not compromise the obstacle avoidance behaviour.

Keywords: Membrane computing, Numerical P systems, Enzymatic numerical P systems, Robot controllers, Simulation, Model checking, Formal verification.

1 Introduction

Over the past two decades, the advancement of *membrane computing* has attracted significant interest and driven innovation in fields like biology, economics, computer science and robotics. This biologically inspired concept defined as a branch of natural computing, has been introduced by Gh. Păun [33] to study parallel, distributed computation inspired by the structure of a living cell. During the last 20 years, membrane computing has achieved both explanatory and applicative purposes, contributing to solving problems from different fields by leveraging biological inspiration. [46] Several variants of membrane systems (P

systems) were proposed [33,35]. The arrangement of the membrane led to different classes of membrane systems. For example, in a *cell-like P system* the membranes are disposed hierarchically. Alternatively, *tissue-like P systems* [45,42] are characterised by the placement of cells at the nodes of a graph, while the environment is modeled as a separate, distinct node. *Spiking neural P systems* represent another category that adopted essential concepts from neurobiology. In a spiking neural P system, *spiking* and *forgetting* rules are applied to a graph of interconnected neurons through synapses [14,20,36,43].
Formal verification allows the checking of system behaviours that cannot be analysed through simulation or testing. It therefore provides additional and often more novel insights into system behaviour. Formal verification has been explored for different categories of P systems. For example, M. Gheorghe et al. [17,18] proposed a method to formally verify the specifications of a P system by first identifying invariants in specification and then use a dedicated tool, called NuSMV to check if the properties as invariants are true or false. The method was applied for some basic categories of P systems with transformation and communication rules. Later on, Ipate et al. presented a method to P system verification using the Spin model checker [26]. Other approaches have been studied [6,27,19,5] for stochastic systems , where the rules of a P system model of a small gene expression system were checked using a dedicated tool, called PRISM (Probabilistic and Symbolic Model Checker) [22,23]. The work was further extended to incorporate the verification of Spiking Neural P systems [20].
A robust method for formal verification of P systems is represented by model checking. Model checking is a formal verification technique used to demonstrate if a system model satisfies a given specification. This involves that each property of the computer system or biological model is expressed in a formal logic and then verified by systematically exploring all possible states of the model. Model checking explores all possible states and execution paths of the system within its model, eliminating the risk of missing rare corner cases. This constitutes a major advantage over classical testing or system simulation [3]. Moreover, when a property is not satisfied, model checkers generate a counterexample trace that illustrates the sequence of steps leading to failure [25].

As mentioned earlier, PRISM is a probabilistic model checker tool. It supports three different types of probabilistic models: discrete time Markov chains (DTMC), Markov decision processes (MDP) and continuous time Markov chains (CTMC) [5]. Accordingly, its use requires an explicitly constructed formal model. This makes the tool less suitable for systems studied primarily through simulation, where such a specification may not be available. Monte Carlo model checker (MC2) [21] is another tool that performs model checking, analysing simulation traces generated by other tools. MC2 is beneficial for systems with large state spaces, where exhaustive exploration is impractical. Both PRISM and MC2 were part of a structured analysis of model checking tools provided by M. Bakir et al. [4].

Membrane systems have also been successfully applied in robotics. A category of cell-like P systems, called *numerical P systems* has been introduced by Gh. Păun and R. Păun in 2006 [34]. A numerical P system evolves numerical variables using a membrane structure, in the form of a tree. The evolutionary process involves the parallel execution of polynomials with integer coefficients, known as production functions. Each membrane contains one or more production functions. The use of numerical variables inside membranes has made numerical P systems, along with their extension, *enzymatic numerical P systems* (ENPS) [32], efficient for robotic control. Unlike traditional numerical P systems, ENPS employ enzyme variables to dynamically control the activation of rules. The natural modeling of reactive behaviours as well as the inherent parallelism that matches real-time requirements make these systems suitable for robotics.
However, designing controllers for autonomous robots that reliably avoid obstacles is a challenging task, as the relationship between sensor readings and motor commands must be both correct and robust across different environmental conditions. Consequently, ensuring that the resulting controller behaves correctly remains highly delicate. The interaction between the membrane computation, the sensor inputs and optimised weight parameters of the controller must be formally proven to guarantee that the intended obstacle avoidance behaviour is actually produced. Moreover, the above mentioned features of ENPS along with the continuous dynamics of the environmental scenarios create significant verification challenges. Aspects like non-determinism, inherent complexity and the large state space of the model make model checking a rigorous yet computationally feasible approach to verify several temporal properties.

In this paper, we investigate the formal verification of obstacle avoidance robot controllers based on enzymatic numerical P systems. The core component of the controllers is an ENPS model that interacts with the environment and takes sensor values at each simulation step, returning the computed wheel speeds of the robot. As the educational robot used in our studies is actuated by two independent motors, the differences between the speeds serve as the primary mechanism governing obstacle avoidance behaviour.
We show how model checking can be applied to study the probabilistic behaviour of such controllers by verifying several properties defined in MC2 using a logic language called Probabilistic Linear-time Temporal Logic (PLTL). While previous work designed the ENPS-based controllers and evaluated their performance by assessing metrics like traversal time, distance or memory consumption [9], this paper applies model checking to formally quantify their key behavioural properties. Each property captures critical correctness requirements and is classified according to different aspects of system operation. We introduce and describe each of these properties and then explore the results produced by MC2 based on the model and different scenarios that are performed in a dedicated simulation environment, providing an effective way of validating the model.

The paper is structured as follows: Section 2 includes the main concepts to be used in this paper. Section 3 provides an overview of the controllers logic and introduces one enzymatic numerical P system model. Section 4 describes the working methodology, including the tools involved. Section 5 explains the properties used for the formal verification. Section 6 illustrates the outputs of the verification process under different scenarios. In the end, Section 7 states the conclusions and the future work in this area.

2 Preliminaries

In this section, we will introduce the principal concepts used in our approach: enzymatic numerical P systems (ENPS), which represent the computational model that is used to build the main logic of the robot controllers and statistical model checking for formal verification.

2.1 Enzymatic Numerical P Systems

As stated before, the primary innovation brought by numerical P systems is the use of numerical values inside compartments. In this way, the compartments contain variables instead of objects. Each variable has an initial value that can be an integer or real number. Another important change is made by the *programs*, that are replacing the traditional *communication rules*. At every time unit, a program from each membrane is executed and the variables are evolved, changing their values. The entire system is governed by a global clock operating in discrete time units.

The enzymatic numerical P systems (ENPS) [34,32] are a variant of numerical P systems that use *enzymes* to control the application of a program during the execution. An enzyme is a variable that allows the execution of more than one production function from each membrane. The enzyme dictates if a program is executed or not, based on the following rule: a program can only be applied (is said to be active) if the corresponding enzyme is present in the necessary amount (has a greater value than the minimum between the variables values involved in that program).

The enzymatic numerical P system (EN P system) is defined by the tuple [8]:

$$\Pi = (m, H, \mu, (Var_1, E_1, Pr_1, Var_1(0)), \ldots, (Var_m, E_m, Pr_m, Var_m(0))) \quad (1)$$

where:

- $m \geq 1$ is degree of the system Π (the number of membranes);
- H is an alphabet of labels;
- μ is membrane structure;

- Var_i is a set of variables from membrane $i, 1 \leq i \leq m$;
- E_i is a set of enzyme variables from membrane $i, 1 \leq i \leq m$;
- $Var_i(0)$ is the initial values of the variables from membrane $i, 1 \leq i \leq m$;
- Pr_i is the set of programs from membrane $i, 1 \leq i \leq m$.

The program $Pr_{l_i,i}$, $1 \leq l_i \leq m_i$ can have two forms, the non-enzymatic, which is identical to the one from the classical numerical P systems and the enzymatic form:

i) non-enzymatic

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

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

ii) enzymatic

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

where e_i is an enzymatic variable from Var_i, $e_i \notin \{x_{1,i}, \ldots, x_{k,i}, v_1, \ldots, v_{m_i}\}$. The program can be applied at time t only if $e_i > min(x_{1,i}(t), \ldots, x_{k,i}(t))$. All the programs that meet this condition in a membrane will be selected to be executed in parallel.

Let us consider that the system applies a program at time $t \geq 0$. Then, let the computed result of

$$q_{l_i,i}(t) = \frac{F_{l_i,i}(x_{1,i}(t), \ldots, x_{k,i}(t))}{\sum_{j=1}^{n_i} c_{j,i}} \tag{4}$$

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

At this point, the previous values of the variables (from time $t-1$) that were present in the production functions are said to be *consumed*, meaning that are set to zero and their new value is the sum of the proportions distributed to variable through the repartition protocols, if they appear in them (on the right side of the program) or remain zero if not.

2.2 Statistical Model Checking

Statistical Model Checking (SMC) is a verification technique in which properties are evaluated through repetitive system simulations. It was introduced by H. Younes in 2002 [31]. As the exploration of all the possible states can lead to state explosion problem, in statistical model checking the approximate level of system correctness is generated using statistical methods. Statistical inference can be approached in two main ways: *hypothesis testing* and *estimation*. Hypothesis testing determines the extent to which observations adhere to a known specification, whilst estimation is employed to infer the probable values of parameters, under the assumption that the data has been randomly sampled from a specified distribution [1].
As mentioned earlier, SMC does not analyse the entire state space and generates an estimated quantification of system correctness. Thus, even if it can not guarantee the absolute correctness, it can produce satisfying results when verifying larger models [11,44].

Statistical Model Checking is a scalable verification paradigm, being useful in verification problems for biological and cyber-physical systems (CPS), as the size of the state space can be difficult to be precisely determined (or may be infinite). Typically, the use of SMC in such cases comprises the following steps [2]:

1. Sampling the potential executions (simulations) of the CPS.
2. Specifying the temporal logic formulas using a property specification language.
3. Verifying that each property is valid on the model.
4. Applying a statistical method to estimate the value of the probability of correctness.

A significant limitation is caused by some temporal formulas for which the probability of satisfaction is minimal. The problem with this formulas, called *rare events*, is that the number of simulations required to obtain a reliable estimate becomes infeasible. E.M. Clarke and P. Zuliani proposed an approach to address this problem using Importance Sampling and Cross-entropy method [12].
A wide range of SMC tools are available, each providing different capabilities in terms of performance and execution speed, when addressing verification models of variable size and complexity. In this paper we will use Monte Carlo Model Checker (MC2) to verify the core component of a CPS, more precisely an obstacle avoidance robot controller of an educational robot.

3 Robot Controllers

In this section we present the operational logic of the robot controllers based on ENPS models. The controllers are purposed to enable the obstacle avoidance functionality of an educational robot, called E-puck [30]. The robot is equipped

with two motors, one for each wheel. Thus, the wheel speed is dictated by the controller. The obstacle avoidance functionality is achieved using the information retrieved by the proximity sensors. Each sensor is associated with a numerical *weight*, which determines its contribution to the control process and is provided as input to the ENPS model, as will be explained when introducing the model instance. In our experiments, a total of six sensors were employed, arranged in a radial configuration with three sensors placed on each side. The robot also has a GPS module installed in the turret slot to accurately measure coordinates.

3.1 Robot Controller Using Static Weights

For a clearer insight into the operational mechanisms of the proposed controller, Algorithm 1 presents the pseudocode representation of the main loop of robot controller using static weight values that are empirically chosen, as defined in [10].

Algorithm 1 Main loop of the robot controller using static weights

```
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 ENPS
6:     leftMotor ← lw
7:     rightMotor ← rw
8: until the end of the road or E-puck goes out of the road
```

One can notice that the above presented pseudocode representation of the algorithm illustrates the way the enzymatic numerical P system (ENPS) exchanges information with the environment. At each simulation step, the sensor values are transmitted to the ENPS using a Python script that implements the logic presented by Algorithm 1 and the communication between the environment and ENPS. In this way, the ENPS takes the sensor values and returns the wheel speeds, denoted by lw and rw (left wheel speed and right wheel speed). The computed values are then assigned to the motor speeds, guiding the robot inside the environment. The tools that facilitate the execution of the ENPS and the simulation of the environment will be introduced in the next section.

3.2 Robot Controller Using Dynamically Assigned Weight Values

The controller variant that was presented above is designed for ENPS models employing static weights within the sensor-driven obstacle avoidance process. While this approach simplifies the controller design, ensuring a predictable behaviour, the adaptability of the system to changing environmental conditions

is significantly reduced. More specifically, during a simulation, the same weight values are maintained regardless of the obstacle density or trajectory complexity, employing laborious manual processes for finding appropriate values, that are, in most cases, empirically chosen.
To address this limitation, an adaptive mechanism of assigning weights during the simulation has been introduced in [7]. The robot controller presented in 3.1 has been tuned using a NSGA-II, a multi-objective optimisation algorithm [15]. More precisely, the robot controller interacts with an instance of NSGA-II with two objective functions that is responsible for generating weight values in concordance with the sensor information retrieved from the simulation environment. Algorithm 2 synthesizes the interaction as follows:

Algorithm 2 Main components of the robot controller using dynamic weights

```
instantiate NSGA-II algorithm
create initial population
repeat
    for i=1 to number_of_sensors do
        sensor_membrane(i) ← value(i)
    run NSGA-II for one generation
    get best individual and update weights
    instantiate new population
    run one simulation step
    read lw, rw from P system
    leftMotor ← lw
    rightMotor ← rw
until the end of the road or E-puck goes out of the road
```

It can be observed that the main components of the controller remain consistent with the implementation illustrated in Algorithm 1, where static weights were employed. In addition, the proximity sensor readings represent the input that is fed to the NSGA-II algorithm. The optimisation process is guided by the objective functions, which evolve a population of candidate weight sets and select, at each control step, the most suitable solution. The resulting weights are subsequently passed to the Enzymatic Numerical P system (ENPS), where membrane programs calculate the wheel velocities, which are then assigned to their corresponding variables (*leftMotor, rightMotor*). The resulting robot behaviour impacts subsequent sensor feedback, forming a continuous *perception–decision–action* loop that facilitates real-time adaptation to dynamic environments.

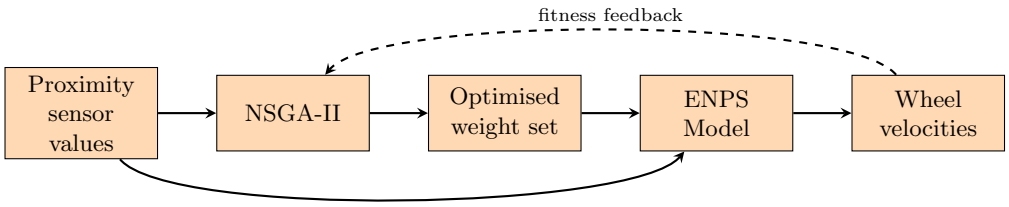


Fig. 1: Dynamic weight optimisation loop integrating NSGA-II with the ENPS-based robot controller

Figure 1 illustrates the information flow of the proposed controller variant. It can be noticed that in the current architecture, the proximity sensor readings serve as input to both the NSGA-II algorithm adaptation, which iteratively constructs the weight set, and the ENPS model, which determines the left and right wheel velocities of the robot.
Although the dynamic assignment of weights demonstrated better performance on certain scenarios and metrics, such as execution time, it also introduced additional complexity into the controller logic. Consequently, the formal verification of the weight assignment requires additional attention to ensure stability, consistency and safe obstacle avoidance. Therefore, this paper also proposes the formal verification of the coordination between the weight parameters and sensor readings.

3.3 Enzymatic Numerical P system Model

The enzymatic numerical P system (ENPS) model is the core component of the robot controllers, dictating the path to be followed by adjusting the wheel speeds. The power of ENPS in robot modeling lies in the ability to represent the robot behaviour using membranes and to precisely control the program execution using enzymes. The integration of enzymes in traditional numerical P systems has led to an optimisation of the computation power of robotic systems. Also, the system variables and enzymes can play a central role in transferring the partial values of the outcomes (i.e. speeds).

In our previous work, we defined a methodology that led us to construct different models in a test-driven manner [9]. The objective of the research is to investigate the relationship between the two central components of the model, governing the robot behaviour. More specifically, the study examines how the proximity sensors, radially distributed around the robot and responsible for obstacle detection and their associated weight parameters are influencing the robot trajectories. As all the developed models exhibited strong performance in passing the testing scenarios, due to the simplicity of the membrane structure, only one model is formally presented in this section. Furthermore, the modifications

incorporated into the other developed instances primarily targeted the naturalness of the robot's movements, without altering the relationship between sensors and weights, which constitutes the central focus of this study. As a result, the properties that are defined in this study may hold the same results for all the developed models employing sensor weights. For more details regarding the other model instances, the reader is invited to consult [9].

The model we present in this section is called *Basic Model with Rotation* and it was created after noticing the limitations of the initial model tested in [10]. Building upon this baseline, an additional compartment which is used to compute the product of the travel speed and the sum of the weights has been added. By incorporating this newly defined compartment, the robot behaviour improved. The equations defining the mathematical determination of *weightLeft* and *weightRight*, as well as their contribution to *leftSpeed* and *rightSpeed*, are presented as follows:

$$weightLeft = \sum_{i=1}^{n} weightLeft_i \cdot prox_i \tag{5}$$

$$weightRight = \sum_{i=1}^{n} weightRight_i \cdot prox_i \tag{6}$$

where $prox_i, 1 \leq i \leq n$ represent the sensor values.

In order to proceed with the equations for the left and right wheel speeds, let us first introduce the following function:

$$f(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Then, the motor speeds, denoted by *leftSpeed* and *rightSpeed* are presented as follows:

$$leftSpeed = cruiseSpeed \cdot weightLeft + f(weightLeft) \cdot cruiseSpeed \tag{8}$$

$$rightSpeed = cruiseSpeed \cdot weightRight + f(weightRight) \cdot cruiseSpeed \tag{9}$$

where *cruiseSpeed* represents the predefined velocity of the robot, when moving straight.

The model instance is defined as follows:

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0))) \tag{10}$$

where:

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

For completeness, the role and interpretation of each variable from the model is clarified below:

- x_{s_l} and x_{s_r} from membrane s represent *leftSpeed* and *rightSpeed*, the sum of the products are accumulated in s after applying the rules from the membranes c_i, where $1 \leq i \leq k$;
- x_{w_l} and x_{w_r} from membrane w represent the *weightLeft* and *weightRight*, computed as described in 5 and 6;
- x_{s_c} from membrane s_c is *cruiseSpeed*;
- variables x_{c_i,s_l} and x_{c_i,s_r} from membrane c_i represent the sensor values multiplied inside membranes s_i, whilst x_{c_i,w_l}, x_{c_i,w_r} are the values of $weightLeft_i$ and $weightRight_i$ derived from membranes w_i;
- each pair of weights, ($weightLeft_i$, $weightRight_i$), is denoted by (x_{w_i,w_l}, x_{w_i,w_r}) and is located in membranes w_i, $1 \leq i \leq k$;
- each proximity sensor, $prox_i$, has an associated membrane s_i, containing a single variable, $x_{s_i,i}$, $1 \leq i \leq k$, holding the last value reported by the sensor;

One can observe the addition of x_{s_l} and x_{s_r} multiplied by zero in the production function from membrane s. The reason for adopting this approach is to represent the reset of the speeds, in order to consume the previous values of *leftSpeed* and *rightSpeed* at each simulation step.

The *weight* membrane w comprises two distinct programs that are responsible of aggregating weight contributions and applying the *cruiseSpeed*. Within the *weight* membrane, the accumulated values of *weightLeft* and *weightRight* are multiplied by *cruiseSpeed*, with a fallback mechanism (adds *cruiseSpeed* if weight

is 0), as outlined in 8 and 9, computing the actual values of the wheel velocities, through the use of enzymes.

It can be noticed that the *cruiseSpeed* membrane s_c incorporates a self-replicating program that preserves the *cruiseSpeed* value and makes it available to the parent compartment w. As highlighted by equations 5 and 6, each proximity sensor has a weighted contribution to *weightLeft* and *weightRight*. This contribution is achieved from the compute membranes $c_i, 1 \leq i \leq n$, where n represents the number of sensors. Furthermore, membranes s_i act like a sensor data distribution layer, distributing copies to the corresponding left and right computation variables (x_{c_i,s_l}, x_{c_i,s_r}) in the parent compartment. An analogous role is assumed by membranes w_i that duplicate each pair of weights, $weightLeft_i$, $weightRight_i$, providing copies (x_{c_i,w_l}, x_{c_i,w_r}) to the respective compute membrane c_i.

4 Methodology and Tools

In order to support the verification process, a number of dedicated tools are employed, each fulfilling a different role, from the simulation environment to model checking. In this section we present an overview of the tools used in this study, highlighting their contribution in the verification process. Rather than deploying the controller logic on the physical version of E-puck, we demonstrate its behaviour through a robotic simulation platform that enables the simulation of complex environments, called Webots [29]. As the core component of the controllers is represented by the ENPS model Π, a tool that is capable of simulating numerical P systems evolutions was needed. Thus, we used PeP [39], a simulator of both classical and enzymatic NPS. Subsequently, we exported the simulation trace produced by the execution of Π and analysed it using Monte Carlo Model Checker (MC2) to formally verify some specified properties. In what follows, we provide additional details on the distinct roles fulfilled by each of the tools employed in this process.

4.1 Webots

Webots [29,13,28] is a robot simulation software that provides a virtual environment for developing and testing robots in customised environments. Special mechanisms called PROTOS are used in conjunction with default shapes to construct complex environments by adding various objects to the scenes. Webots provides various accessories that can be mapped to robots, such as GPS, light sensors or LED. During the simulation phase, we attached a GPS to the turret slot of the robot in order to pass the system coordinates to the controller.
A simulation scenario is called *world*, defined as a *.wbt* file, containing the definition and placement details for all the object from the scene. The *.wbt* can be edited in order to extend a scene.
A virtual representation of E-puck, equipped with 6 sensors, replicating the main functionalities of the original robot has been involved in each scenario. The robot

controller is editable in the simulation interface and supports multiple programming languages, such as C, Java, Matlab or Python. Our developed obstacle avoidance controller is a Python file that exchanged data with the ENPS model at each simulation step.
For additional details regarding the integration of Webots in a similar experimental setup, the reader is invited to consult [10].

4.2 PeP

As already mentioned, the robot controller takes the wheel speed values from the ENPS model Π at each simulation step. This process entails the use of a dedicated tool for simulating ENPS.
PeP is an open-source tool developed by A. Florea and C. Buiu [39]. It uses *.pep* input files to model NPS and ENPS instances, by defining the membrane structure, variables, enzymes and rules. PeP provides options for deterministic or randomised program selection and supports generic mathematical functions in production and distribution rules. The tool provides detailed execution traces, illustrating the evolution of variables and enzymes. It can be executed for a predefined number of steps and facilitates options for debug and export.
As the tool demonstrates strong applicability for the experimental evaluation of ENPS models, we used it iteratively inside the main loop of the controller. At the end of each simulation, we exported simulation traces as a CSV file, containing the variable values at each step.
An important advantage of the tool lies in its capability to generate traces in an organised form, representing a foundation for subsequent statistical formal verification using a model checker.

4.3 MC2 Model Checker

The Monte Carlo Model Checker (MC2) [21] was developed as a tool for statistical model checking based on simulation. The tool has proven effective results for complex models where exact model checking would be infeasible due to the state-space explosion problem.
MC2 facilitates statistical model checking of simulation traces and supports parallel execution. A notable advantage of the tool is that it does not require a specific modeling language; instead, it is based on simulation traces generated by external tools for both stochastic and deterministic models. The tool is developed in Java and distributed as a *.jar* file.
Properties in MC2 are defined using a logic language called Probabilistic Linear-time Temporal Logic (PLTL). The basic unit of this language is denoted by the Atomic Proposition (AP) which is included in properties without temporal operators. According to the user manual [41], there are five temporal operators defined as follows:

- **Next (X)**: the property must hold true in the next time point;
- **Globally (G)**: the property must hold true always in the future;

- **Finally (F)**: the property must hold true sometime in the future;
- **Until (U)**: the first property must hold true until the second property holds true;
- **Release (R)**: the second property can only ever not hold true if the first property becomes true.

In our approach, we used PeP to export the simulation traces and after minimal format adjustments we passed the file to the model checker. At the same time, we defined several queries written in PLTL into a separate query definition file. Each query has a possible output of true, if the specified property is satisfied in the simulation traces, or false if not. MC2 was selected over other model checking tools (such as PRISM) primarily because it eliminates the need to construct an explicit formal model in a dedicated language. Instead, it operates directly on execution traces generated by external tools. This aligns with our workflow, where the ENPS model is simulated using PeP and the resulting traces are exported for verification. MC2 operates via command-line interface executed from the directory containing the *.jar* file, where we also placed the simulation traces and the file containing the queries. In what follows, we will provide more details about the queries definitions.

5 Formal Verification Properties

In this section, several properties are defined using MC2. The main goal of the properties is to verify the critical correctness requirements of the ENPS model Π that is the core component of the obstacle avoidance robot controllers. By analysing the formal definition of model Π, the expected behaviour at each computation step can be formally characterised and expressed using PLTL properties. In addition, model Π serves as the analytical foundation from which each property is derived, not as a direct input to the verification tool. The Boolean nature of the expected outcomes (false for conditions that must never be reached, true for conditions that must be reachable) reflects the evaluation of whether the production rules of the ENPS model produce the expected results. Unlike exhaustive model checking approaches, MC2 allows to obtain confidence bounds on property violations without compromising the computational costs. Each property is expressed using Probabilistic Linear-time Temporal Logic (PLTL), employing temporal operators.

To some extent, the properties we present can be classified into the following categories:

- *liveness* properties: ensure that the system does not halt. In our approach, this type of properties verifies that the controller successfully transmits values for the wheel speeds at each simulation step and the system maintains its progress.
- *safety* properties: specify conditions that must always hold true during system execution. In our approach, these properties are related to the controller

capacity to react accordingly to the environmental conditions, i.e. to detect and avoid obstacles. In terms of system states, these properties are specified to guarantee that the verified system does not reach undesirable states, such as getting stuck in an obstacle.
- *invariant* properties: verify that certain behavioural characteristics of the systems remain consistent during the system execution. In our approach, this type of properties ensures that unnatural behaviours, such as spinning moves, do not occur during the simulation.

Observing the below presented properties through simulation is impractical because they specify constraints over multiple states and time points that may not be evident from observing individual execution traces. Thus, applying model checking enables systematic verification of these complex temporal relationships that would otherwise require manual inspection across numerous scenarios. In what follows, we provide both the formal PLTL specification and the description of what each property ensures about controller behaviour, along with the explained possible outputs.

Section 5.1 defines properties that operate on the input provided by the simulation environment to the ENPS (the proximity sensor readings, *s1-s6*) and on the output of the ENPS model (the wheel velocities, *lw* and *rw*). Consequently, they verify the correctness of the sensor-motor mapping regardless of how the weights are obtained. Therefore, they apply equally to both the static and the dynamic controllers that are described in Section 3.

Remark 1. For clarity and simplicity, we use the actual variable names from the model implementation rather than introducing the variables defined in the formal definition presented in 3.3. For example, the wheel speeds are denoted by *lw* and *rw* instead of membrane variables x_{s_l} and x_{s_r}. For the sensor values, we use $s_i, 1 \leq i \leq 6$ instead of $x_{s_v,i}, 1 \leq i \leq 6$.

5.1 Common Properties

Property 1 (Liveness). $P =?[F([lw] \neq [rw])]$

Description. This property verifies that the controller is capable of producing asymmetric wheel speeds at some point during the simulation, checking the necessary condition for the robot to turn and avoid obstacles. In model Π, *lw* and *rw* are computed independently by the Weight membrane *w* through two separate rules, each accumulating a distinct weighted sum, *weightLeft* and *weightRight*. The temporal operator "Finally" (F) means that the condition must hold at some point in the future along at least one execution path. By analysing the physical structure of the robot, one can notice that the robot will turn if and only if the wheel speeds will take different values during the simulation. In the context of an obstacle avoidance controller, it is essential to ensure that the wheel speeds are adjusted properly, based on inputs from the environment. This type of property is often encountered in automotive area when verifying the

turning dynamics of the prototype, where asymmetric wheel speeds are expected under certain circumstances.
Output. The property will hold true if the wheel speeds will differ at some points during the system execution, ensuring that the controller is able to turn the robot in a direction. If the wheel speeds will maintain equal values during the entire simulation, the model checker will return zero (false), as this means that the robot only moves straight ahead.

Property 2 (Liveness). $P =?[F(([lw] \neq [rw]) \wedge X([lw] = [rw]))]$

Description. This property verifies that after successfully avoiding an obstacle, the robot eventually reorients itself to a centered trajectory. When the robot avoids an obstacle and sensor values return toward zero, the membranes associated to each proximity sensor forward weighted products to the Weight membrane, causing *weightLeft* and *weightRight* to converge. When both aggregate values evaluate to zero, the fallback mechanism defined in 8 and 9 and implemented through rules in Pr_w produces identical outputs for both wheel speeds, *lw,rw*. Since the Speed membrane resets both wheel speed values at every step, symmetry is restored within a single step, ensuring that no residual values from previous configurations affect subsequent computations. This analysis is useful for identifying undesirable motion patterns, including potential spinning or circular movements, which could compromise the stability and reliability of the navigation. The scope of this property is to determine whether the robot has the ability to return to a stable, centered trajectory after obstacle avoidance.
Output. This property will hold true if there exist at least once two consecutive steps in the simulation traces where the speeds are initially different and then they become equal, indicating that the robot follows a straight trajectory after performing an obstacle avoidance manoeuvre.

Property 3 (Liveness).
$P =?[F([s1] > 0 \vee [s2] > 0 \vee [s3] > 0 \vee [s4] > 0 \vee [s5] > 0 \vee [s6] > 0)]$

Description. This property verifies that model Π receives the sensor values from the testing environment. Although the primary aim of system execution verification focuses on the analysis of output values, it is also important to consider and verify the system's interactions with its connected components. As stated in Algorithm 1, in our particular setup, the main loop of the robot controller comprises the exchange of information between the ENPS model and the environment. The desired behaviour implies that Π model takes the sensor values from the environment, computes and returns the wheel speeds at each simulation step.
Output. The property will hold true if at least one sensor will be triggered at some point in the future. As the absence of nearby obstacles is denoted by $s_i = 0, \forall i \in \{1, \ldots, 6\}$, the use of the Finally (F) operator is appropriate.

Property 4 (Safety). $P =?[G([lw] \neq 0 \wedge [rw] \neq 0)]$

Description. This property verifies that the wheel speeds remain different from zero throughout the simulation, ensuring continuous motor output at every computation step. As it was defined, the Π model computes the values of *lw,rw* wheel speeds that are used in the main loop of the robot controller. Once the simulation starts, the model must return values that move the robot inside the testing environment. The computation is guided by the sensor values and the weights, as stated in Pr_w compartment of the ENPS model. When no obstacle is detected, the *cruiseSpeed* variable value that is persisted by a dedicated program, ensures that both wheel speeds default to the cruise speed value. The "Globally" temporal operator (G) denotes that the property must hold true during the entire execution.
Output. The property will hold true if *lw* and *rw* will be different from zero and false if at least one wheel speed becomes zero during the simulation.

Property 5 (Safety).

$$P = ?[F((\sum_{i=1}^{3}[s_i] > 0) \wedge (\sum_{i=4}^{6}[s_i] = 0) \wedge X(X(X([lw] \leq [rw])))]$$

Description. This property is designed to detect a situation that is not intended during the system execution. An immediate analysis of the robot dynamics indicates that when the proximity sensors detect an obstacle on the left side, the ENPS model Π *must* adjust the wheel speeds by producing *lw* strictly greater than *rw*. The property verifies whether there exists at least one state where the sensors placed on the left side are triggered ($s1 + s2 + s3 > 0$), indicating that an obstacle is detected on the left of the robot and *lw* will become less than or equal to *rw* after two simulation steps, which would contradict the intended avoidance behaviour.
One can notice that safety properties were defined as a type of properties that must always hold true during the entire execution of the system. Consequently, the way this definition has been presented may suggest the use of the Globally (G) temporal operator. As it was defined, the G temporal operator asserts that a property must hold at every single state throughout the execution. By using it individually, the G operator operates on individual states rather than sequences of states and does not have the capability to capture consecutive steps or to limit the number of steps over which a property should return true. In this way, the Next operator (X) can be used to verify that a multiple step sequence occurs during execution. By definition, *X(property)* verifies that the property must hold true in the next simulation step. Additionally, *X(X(X(property)))* will determine whether the property is true in the third simulation step (at moment *t+3*). The use of the Next operator in this manner can chain together n-consecutive steps, potentially combined with G or F operators.
Our experiments revealed that having the Globally operator G combined with Next (X) using logical Boolean operators will return false negative outcomes when there are less than n-consecutive steps remaining in the simulation. For example, if the objective is to verify that after detecting an obstacle on the right side, the robot will turn left during the next 3 steps, but just two steps are remaining in the simulation trace, the property will hold false. In order to

avoid these cases, we negate the consequent conditions that must hold true during the next steps and use the Finally (F) operator. This approach avoids the fundamental problem with G enforcing an universal quantification over all states.

With respect to the fourth property definition, the following observations may be drawn:

- multiple conditions are linked using "logical and" (conjunction) operator;
- the Finally operator F incorporates the entire property definition, meaning the returned result will be true if the conditions are met at some point during the simulation;
- by analysing the definition of model Π, it can be observed that the controller reacts to environmental inputs over a three-step interval. During these steps, the sensor variables are assigned and distributed, the *weightRight* and *weightLeft* values are computed and the motor speeds are determined.

The Finally operator is used to verify whether such situations occur at least once during the simulation. This representation of the property enables the detection of undesired behaviour during simulation, while avoiding the case where the three-step condition cannot be satisfied due to the proximity to the end of the trace (in that case, will correctly return zero, as we cannot determine an undesired behaviour without all the necessary steps).
Output. The property will hold true if sometime in the future the sensors on the left side are triggered (and not the others) and the left wheel speed will remain less than or equal to the right wheel speed after two steps. The desired outcome of this property is false, as it checks the presence of an unwanted behaviour.

Property 6 (Safety).
$P =?[F((\sum_{i=1}^{3}[s_i] = 0) \wedge (\sum_{i=4}^{6}[s_i] > 0) \wedge X(X(X([lw] \geq [rw]))))]$

Description. This property verifies if there exists at least one case when an obstacle is detected on the right side of the robot and the left wheel speed remains greater than or equal to the right wheel speed after two steps. The property implements the above described logic of the fourth property, with the remark that in this case the robot reaction for obstacles positioned on the right side is verified.
Output. The property will hold true if there exists at least one point in the execution where the right-sided sensors are triggered and the right wheel will remain less than or equal to the right wheel speed after two consecutive steps. As in the case of the previous property, the desired result of this property is false.

Property 7 (Invariant). $P =?[F(([lw] > [rw]) \wedge X([lw] < [rw]))]$

Description. This property verifies if the robot exhibits sudden directional changes during the system execution. As in the previous properties, the Next (X) operator is used inside the Finally block, meaning that the property verifies

the presence of two consecutive steps satisfying the conditions in the simulation trace. The ENPS model Π returns the wheel speeds at each simulation step, based on the received environment inputs. Consecutive values of lw and rw are determined by *weightLeft* and *weightRight* variables within the Weight membrane w. A simulation step constitutes a minimal unit of measurement in the context of a complete execution, being imperceptible in the visual representation of the scene. It is also worth noting that the infrared proximity sensors of E-puck return non-zero readings when an obstacle is within a short detection range of approximately 4 cm. As a result, simultaneous activation of sensors on opposing sides is largely restricted to dense or dynamically changing environments with multiple obstacles. Consistent with our test suite where no such configuration arises, we can assert that in typical obstacle avoidance scenarios, the verified model should not produce two consecutive mutually exclusive outcomes, namely one with the left wheel rotating faster than the right, and the next one with the right exceeding the left one.

Output. The property will hold true if there exists at least one case of abrupt directional changes in the simulation trace. As a consistent behaviour implies the robot maintaining its current trajectory without successive abrupt directional changes, the expected output of this property is false in the typical scenarios for which the controller was designed, such as road tests, corridor navigation and arena environments where the robot does not navigate through passages narrow enough to simultaneously trigger sensors on both sides.

5.2 Dynamic Weight Assignment

As stated in 3.2, the integration of NSGA-II as an optimisation of the robot controller increases the complexity of the proposed solution. According to this approach, the sensor weights are no longer maintained as fixed parameters. They become dynamic variables whose values continuously evolve according to the current navigation context, which is encoded through sensor values serving as inputs to the optimisation algorithm. However, as the weight values are no longer empirically chosen static parameters, inappropriate weight combinations or abrupt transitions may occur during the simulation. Consequently, additional attention must be dedicated to the formal verification of the weight values in order to ensure stability and safe obstacle avoidance performance. In this regard, several additional properties are proposed to complement the verification conditions previously defined.

Property 8 (Liveness). $P =?[F([weightLeft] \neq [weightRight])]$

Description. This property verifies that the weight aggregation mechanism of the ENPS model is capable of producing asymmetric total weights across the left and right motor speeds. It aims to find if there exists at least one state in which the accumulated value of *weightLeft* differs from that of *weightRight*. It is worth mentioning that the aggregate weight computation involves the sensor

readings, values different from zero thereby denoting the presence of an obstacle. As can be observed from the membrane structure of model Π, there are 6 Compute membranes ($c_i; 1 \leq i \leq 6$). Each of these membranes contributes to *weightLeft* and *weightRight*, as described by the rules from Pr_{c_i}, $1 \leq i \leq 6$. Since the accumulation channels operate independently, the model permits, but does not guarantee asymmetric outputs. Consequently, the property verifies that this asymmetry is indeed produced during the execution.

Output. This property will hold true if the aggregated weights will differ at least once during the simulation, showing that the model is structurally capable of generating turning behaviour. A result of false would imply that the model always produces symmetric outputs.

Property 9 (Safety). $P =?[F((\sum_{i=1}^{3}[s_i] > 0) \wedge (\sum_{i=4}^{6}[s_i] = 0) \wedge X(X([weightLeft] \leq [weightRight])))]$

Description. This property validates the correctness of the dynamically generated weights relative to the proximity sensor values. Specifically, it checks if there exists at least one state where the left side proximity sensors *(s1,s2,s3)* report the presence of an obstacle, while no obstacle is detected by the right side sensors and yet the aggregate *weightLeft* is less than or equal to *weightRight* in the subsequent step.

As the robot controller is guided by proximity sensors with associated weights, the operating principle assumes that the weight values influence the obstacle avoidance behaviour. Accordingly, when an obstacle is detected on the left side, the expected behaviour is characterised by positive values for the left weights associated with *(s1,s2,s3)*, exceeding the corresponding right weights. Similarly, an analogous property can be defined for obstacles that are located on the right side by modifying the verification logic to target bigger positive values for the right weights that are associated with *(s4,s5,s6)*, as resulted from equations 5 and 6.

Output. This property will hold true if only *(s1,s2,s3)* sensors detect an obstacle, but the aggregate *weightLeft* is less than or equal to *weightRight*. Thus, the expected output of this property is false, as the optimised weight set must produce $weightLeft > weightRight$, when only the left sensors are active. A positive result would indicate that the dynamically assigned weight values do not yield an adequate avoidance response for left-sided obstacle detections.

Property 10 (Safety). $P =?[F([s1] > 0 \wedge (\sum_{i=2}^{6}[s_i] = 0) \wedge ([weightLeft1] < 0) \wedge X(X(X([lw] \leq [rw]))))]$

Description. This property provides verification of the dynamic weight assignment at the sensor level. A Braitenberg vehicle is a concept from robotics that demonstrates how simple mechanisms incorporating sensors and motors can exhibit intelligent behaviour through mappings between sensory information and motor commands [37]. Inspired by this principle, the scope of this property is

to check whether there exists a state where sensor 1 detects an obstacle on the left side, but the robot does not avoid by turning right, even if no other sensor detects an obstacle, because NSGA-II has generated a negative weight for sensor 1 contribution to the left wheel. In general, left-side obstacles should induce weight configurations that favour $lw > rw$, causing the robot to steer to the right and avoid the obstacle, while right-side obstacles should produce weight configurations that favour $rw > lw$. Using the same verification logic as this property, analogous simulation checks can be conducted for each of the remaining proximity sensors.

Output. This property will hold true if there exists at least one point in the execution trace where NSGA-II has generated a negative value for *weightLeft1* that is associated with the detection of an obstacle placed on the left side and the sign error has propagated through model Π to produce $lw \leq rw$. Consequently, the expected result of this property is false.

Property 11 (Safety).

$$P =?[F(([s1] > 0) \wedge ([weightLeft1] = 0 \wedge [weightRight1] = 0))]$$

Description. This property verifies whether the NSGA-II algorithm can produce null weights for proximity sensor s1, while the sensor is activated. Specifically, it queries whether there exists a reachable state where *s1* reports a positive reading, indicating an obstacle in the front left region of the robot and the associated weights, *weightLeft1* and *weightRight1* are simultaneously zero. By looking at equations 5 and 6 and their contribution to 8 and 9, it can be noticed that a correct response to a left-side obstacle implies $weightLeft_i > weightRight_i$, $i \in \{1, 2, 3\}$, whereas effective avoidance of a right-side obstacle requires $weightRight_i > weightLeft_i, i \in \{4, 5, 6\}$. Therefore, with no contribution from the associated weights, the robot will move straight ahead, "ignoring" the reported sensor value.

Output. The expected result of this property is 0, meaning that no such state should be reachable under the weight configurations produced by NSGA-II. In this regard, the property can be generalised for all the remaining sensors as the weight generation process is directly targeted, ensuring that the dynamic assignment preserves the desired relationship between sensors placement and the corresponding robot behaviour.

Property 12 (Invariant).

$$P =?[F(([weightLeft] < [weightRight]) \wedge X([lw] \geq [rw]))]$$

Description. This property checks whether there exists any reachable execution step where *weightLeft* is strictly less than *weightRight* at one step and the left wheel velocity *lw* is greater than or equal to the right wheel velocity *rw* at the next step. It aims to validate the existing mapping between the aggregate weights and wheel velocities.

Output. Similarly to some of the previously introduced properties, a value of false for this property indicates the correct functioning of the ENPS based robot controller, as the radial arrangement of the proximity sensors implies that a lower

aggregate left weight must always produce a strictly lower left wheel speed.

Property 13 (Invariant).
$P =?[F(([weightLeft] = [weightRight]) \wedge X([lw]! = [rw]))]$

Description. This property verifies that equal aggregate weights must always produce equal motor speeds. As mentioned earlier, the simulation traces have highlighted that the absence of nearby obstacles results in computed aggregate weights that remain zero, due to the fact that none of the sensors reports positive values. In such scenarios, the expected behaviour is that the system will produce equal motor speeds, allowing the robot to move in a straight line.
Output. This property will hold true if there exists at least one sequence of two simulation steps where aggregate weights produce different motor speeds. The expected value of this property is thereby false, validating the determinism and symmetry of the speed computation with respect to the aggregate weights.

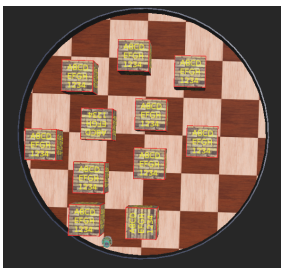
Property 14 (Invariant).
$P =?[F(([weightLeft] > [weightRight]) \wedge X([weightRight] > [weightLeft]))]$

Description. This property verifies the presence of abrupt changes in weight aggregates. As already mentioned, the weight values are dynamically generated in accordance with the proximity sensor values. Thus, it can be noticed that this property is structurally similar to Property 6. However, whereas Property 6 focuses on detecting abrupt velocity variation, the dynamic generation of weights requires additional verification to ensure that the aggregated weight values do not produce abrupt changes themselves.
Output. This property will hold true if abrupt oscillation of aggregate weights is detected. When considered alongside Property 6, the result of this property may suggest the necessity for further tuning of the weight optimisation objectives.

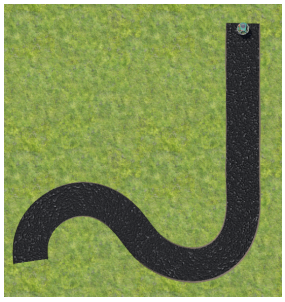
6 Results

This section presents the verification results based on the previously described properties. Several simulation scenarios were developed using Webots graphical interface. In what follows, we briefly introduce the scenarios designed to challenge the obstacle avoidance robot controllers based on the ENPS model Π.
Initially, we generated multiple curved roads based on spines produced using a search-based framework for autonomous systems testing [24,16]. More details regarding the construction of the road tests can be consulted in the previous work [38]. The edges of each road are treated by the robot as obstacles. Therefore, the controller has to guide the robot along the entire road while maintaining a trajectory free of collisions with respect to the boundaries.
Another test scenario consists of situating the robot in circular arenas containing a variable number of obstacles, each being represented within the environment using Webots default shape objects. The arena radius as well as the number of

obstacles are configurable parameters. Given the arrangement of obstacles, the robot's objective is to navigate through the arena without colliding any of them. This scenario can be executed for a predetermined time interval, or until the robot navigates through the obstacles and reaches the boundary of the arena. In our approach, we placed 11 boxes and the simulation is running until the robot escapes the boxes area and reaches the margin of the arena.



Circular arena with obstacles



Curved road

To support reproducibility, the ENPS model, all queries, simulation traces and controller implementations used in this study are available at [40].

Before discussing the results, we emphasize that the verification process was structured into the following stages:

- We simulated the above described scenarios and derived the execution traces of the ENPS model Π. In the case of curved roads, we extracted traces from 20 roads with different characteristics, whose complexity varied according to the number of curves and their positioning. From the arena scenarios, 5 simulation traces were collected by systematically varying the robot's starting position.
- For the robot controller using dynamically assigned weights, we ran each simulation 10 times, as the optimisation algorithm produces non-deterministic values
- For each testing scenario, the full execution trace was collected, covering the entire simulation from the start until the robot successfully completed the road or reached the boundary of the arena. Each trace contains thousands of entries, where each entry corresponds to a single simulation step. The traces were exported in CSV format by PeP, containing the values of each membrane variable during the simulation. Each row of the CSV corresponds to a simulation step
- The simulation outputs were translated into a data format accepted by the MC2 verification tool.

– Based on the query definition file containing the above presented properties and the simulation output file we performed model checking using MC2 from the Windows command line. MC2 was executed in stochastic mode, evaluating each PLTL property over the set containing 25 simulation traces (20 curved road traces and 5 circular arena traces).

Table 1 presents the content of the query definition file along with the associated results of each property:

Table 1: Verification results

ID	**PLTL Formula**	**Result**
P1	$P =?[F([lw] \neq [rw])]$	1.0
P2	$P =?[F(([lw] \neq [rw]) \wedge X([lw] = [rw]))]$	1.0
P3	$P =?[F([s1] > 0 \vee [s2] > 0 \vee [s3] > 0 \vee [s4] > 0 \vee [s5] > 0 \vee [s6] > 0)]$	1.0
P4	$P =?[G([lw] \neq 0 \wedge [rw] \neq 0)]$	1.0
P5	$P =?[F((\sum_{i=1}^{3}[s_i] > 0) \wedge (\sum_{i=4}^{6}[s_i] = 0) \wedge X(X(X([lw] \leq [rw]))))]$	0.0
P6	$P =?[F((\sum_{i=1}^{3}[s_i] = 0) \wedge (\sum_{i=4}^{6}[s_i] > 0) \wedge X(X(X([lw] \geq [rw]))))]$	0.0
P7	$P =?[F(([lw] > [rw]) \wedge X([lw] < [rw]))]$	0.0
P8	$P =?[F([weightLeft] \neq [weightRight])]$	1.0
P9	$P =?[F(((\sum_{i=1}^{3}[s_i] > 0) \wedge (\sum_{i=4}^{6}[s_i] = 0)) \wedge X(X([weightLeft] \leq [weightRight]))))]$	0.0
P10	$P =?[F(([s1] > 0) \wedge (\sum_{i=2}^{6}[s_i] = 0) \wedge ([weightLeft1] < 0) \wedge X(X(X([lw] \leq [rw])))))]$	0.0
P11	$P =?[F(([s1] > 0) \wedge ([weightLeft1] = 0 \wedge [weightRight1] = 0))]$	0.0
P12	$P =?[F(([weightLeft] < [weightRight]) \wedge X([lw] \geq [rw]))]$	0.0
P13	$P =?[F(([weightLeft] = [weightRight]) \wedge X([lw]! = [rw]))]$	0.0
P14	$P =?[F(([weightLeft] > [weightRight]) \wedge X([weightRight] > [weightLeft]))]$	0.0

Table 1 illustrates the output from MC2 for the model checking of robot controllers based on model Π. The results of 0.0 and 1.0 indicate that each property was either satisfied or violated consistently across all traces. This uniformity is expected, as the verified properties target the structural correctness of the ENPS model itself rather than scenario-dependent behaviour. Specifically, they evaluate the rules governing weight aggregation, sensor-motor mapping and wheel velocity computation. The query results validate that model Π ensures the robot movement while simultaneously exhibiting the capability to adjust the wheel speeds as dictated by the environmental dynamics (transmitted as sensor values with associated static/dynamic weights) and computed using the kinetic rules implemented as membrane programs. All the temporal properties evaluated through MC2 model checking returned the desired outcome, demonstrating alignment between the observed system behaviour and the expected functional requirements. While the consistent results across all tested scenarios support the validity of the findings, statistical model checking estimates property satisfaction

based on sampled executions and does not provide guarantees beyond the use case scenarios provided and the diversity of the collected trace set.

7 Conclusions and Future Work

This paper proposes a practical approach for verifying two similar obstacle avoidance robot controllers based on ENPS, one using static weights and the other one taking dynamic weights returned by an integrated multi-objective optimisation algorithm. The verification process involved the use of several tools, with their selection and roles described within this paper. The use of statistical model checking, rather than exhaustive testing, enables the derivation of confidence bounds without compromising computational efficiency.

The results of our experiments proved that the ENPS model correctly implements the desired control logic using both empirical static weights and the weight configurations produced by the NSGA-II adaptation. Furthermore, several properties demonstrated that, within the simulation traces, the integration of the multi-objective optimisation algorithm within the main logic of the controller does not introduce inconsistencies in the mapping from proximity sensor readings to wheel velocities, thus serving as an argument in validating the generated weight sets.

As future work, the proposed approach will be extended with properties that depend not only on the ENPS model but also on the execution scenario. Unlike the properties considered in this study, which verify model correctness by producing deterministic binary outcomes, the new properties will be evaluated over collections of simulation traces, enabling the estimation of probabilities associated with scenario-specific events. For example, given a set of traces collected from different road tests, MC2 can estimate the probability that the robot first detects an obstacle on its right side.

It is worth mentioning that ENPS was adopted in this study because it constitutes the core modelling component of the robot controller. However, the above presented experimental stages can lead to a generalised approach for the formal verification of different numerical P systems variants (e.g., employing Boolean conditions) using model checking.

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Transactions on Modeling and Computer Simulation (TOMACS) **28**(1), 1–39 (2018)
2. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
3. Bakir, M.E., Konur, S., Gheorghe, M., Niculescu, I., Ipate, F.: High performance simulations of kernel P systems. In: 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS). pp. 409–412. IEEE (2014)

4. Bakir, M.E., Gheorghe, M., Konur, S., Stannett, M.: Comparative analysis of statistical model checking tools. In: International Conference on Membrane Computing. pp. 119–135. Springer (2016)
5. Bernardini, F., Gheorghe, M., Romero-Campero, F.J., Walkinshaw, N.: A hybrid approach to modeling biological systems. In: International Workshop on Membrane Computing. pp. 138–159. Springer (2007)
6. Blakes, J., Twycross, J., Konur, S., Romero-Campero, F.J., Krasnogor, N., Gheorghe, M.: Infobiotics Workbench: A P systems Based Tool for Systems and Synthetic Biology, pp. 1–41. Springer International Publishing, Cham (2014)
7. Bobe, R.T., Gheorghe, M., Ipate, F.: Submitted: Sensor-driven evolutionary optimisation of mobile robot controllers using enzymatic numerical P systems (2026), https://doi.org/10.13140/RG.2.2.23913.04969, researchGate [Preprint]
8. Bobe, R.T., Gheorghe, M., Ipate, F., Niculescu, I.M.: Numerical P systems: Variants and applications (2024)
9. Bobe, R.T., Gheorghe, M., Ipate, F., Niculescu, I.M.: A test-driven methodology for designing robot controllers and simulators using enzymatic numerical P system models (2024)
10. Bobe, R.T., Ipate, F., Niculescu, I.M.: Modelling and search-based testing of robot controllers using enzymatic numerical P systems. In: Cheval, H., Leuştean, L., Sipoş, A. (eds.) Proceedings 7th Symposium on Working Formal Methods, Bucharest, Romania, 21-22 September 2023. Electronic Proceedings in Theoretical Computer Science, vol. 389, pp. 1–10. Open Publishing Association (2023). https://doi.org/10.4204/EPTCS.389.1
11. Buchholz, P.: A new approach combining simulation and randomization for the analysis of large continuous time markov chains. ACM Transactions on Modeling and Computer Simulation (TOMACS) **8**(2), 194–222 (1998)
12. Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: International symposium on automated technology for verification and analysis. pp. 1–12. Springer (2011)
13. Couceiro, M.S., Vargas, P.A., Rocha, R.P.: Bridging the reality gap between the webots simulator and e-puck robots. Robotics and Autonomous Systems **62**(10), 1549–1567 (2014). https://doi.org/10.1016/j.robot.2014.05.007
14. de la Cruz, R.T.A., Cabarle, F.G.C., Macababayao, I.C.H., Adorna, H.N., Zeng, X.: Homogeneous spiking neural P systems with structural plasticity. Journal of Membrane Computing **3**(1), 10–21 (2021)
15. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
16. Gambi, A., Jahangirova, G., Riccio, V., Zampetti, F.: Sbst tool competition 2022. In: Proceedings of the 15th Workshop on Search-Based Software Testing. pp. 25–32 (2022). https://doi.org/10.1145/3526072.3527538
17. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., Lefticaru, R.: Kernel P systems: from modelling to verification and testing. Theoretical Computer Science **724**, 45–60 (2018)
18. Gheorghe, M., Ipate, F., Lefticaru, R., Dragomir, C.: An integrated approach to P systems formal verification. In: International Conference on Membrane Computing. pp. 226–239. Springer (2010)
19. Gheorghe, M., Konur, S., Ipate, F.: Kernel P systems and stochastic P systems for modelling and formal verification of genetic logic gates. In: Advances in Unconventional Computing: Volume 1: Theory, pp. 661–675. Springer (2016)

20. Gheorghe, M., Lefticaru, R., Konur, S., Niculescu, I.M., Adorna, H.N.: Spiking neural P systems: matrix representation and formal verification. Journal of Membrane Computing **3**(2), 133–148 (2021)
21. Gilbert, D., Donaldson, R.: A monte carlo model checker for probabilistic ltl with numerical constraints (2008)
22. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoretical Computer Science **391**(3), 239–257 (2008)
23. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: Prism: A tool for automatic verification of probabilistic systems. In: International conference on tools and algorithms for the construction and analysis of systems. pp. 441–444. Springer (2006)
24. Humeniuk, D., Antoniol, G., Khomh, F.: Ambiegen tool at the sbst 2022 tool competition. In: Proceedings of the 15th Workshop on Search-Based Software Testing. pp. 43–46 (2022). https://doi.org/10.1145/3526072.3527531
25. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and reasoning about systems. Cambridge university press (2004)
26. Ipate, F., Lefticaru, R., Tudose, C.: Formal verification of P systems using spin. International Journal of Foundations of Computer Science **22**(01), 133–142 (2011)
27. Konur, S., Gheorghe, M.: A property-driven methodology for formal analysis of synthetic biology systems. IEEE/ACM Transactions on Computational Biology and Bioinformatics **12**(2), 360–371 (2014)
28. Ma'arif, A., Nuryono, A.A., et al.: Vision-based line following robot in webots. In: 2020 FORTEI-International Conference on Electrical Engineering (FORTEI-ICEE). pp. 24–28. IEEE (2020)
29. Michel, O.: Cyberbotics ltd. webots™: professional mobile robot simulation. International Journal of Advanced Robotic Systems **1**(1), 5 (2004). https://doi.org/10.5772/5618
30. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.C., Floreano, D., Martinoli, A.: The e-puck, a robot designed for education in engineering. In: Proceedings of the 9th conference on autonomous robot systems and competitions. pp. 59–65. IPCB: Instituto Politécnico de Castelo Branco (2009)
31. Musliner, D.J., Younes, H.L.: Probabilistic plan verification through acceptance sampling
32. Pavel, A., Arsene, O., 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). pp. 1331–1336. IEEE (2010). https://doi.org/10.1109/BICTA.2010.5645071
33. Păun, G.: Membrane computing: an introduction. Springer Science & Business Media (2002). https://doi.org/10.1007/978-3-642-56196-2
34. Păun, G., Păun, R.: Membrane computing and economics: Numerical P systems. Fundamenta Informaticae **73**(1-2), 213–227 (2006), http://content.iospress.com/articles/fundamenta-informaticae/fi73-1-2-20
35. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
36. Qiu, C., Xue, J., Liu, X., Li, Q.: Deep dynamic spiking neural P systems with applications in organ segmentation. Journal of Membrane Computing **4**(4), 329–340 (2022)
37. Shaikh, D., Rañó, I.: Braitenberg vehicles as computational tools for research in neuroscience. Frontiers in bioengineering and biotechnology **8**, 565963 (2020)

38. Traian Bobe, R., Gheorghe, M., Ipate, F., Mihai Niculescu, I.: A test-driven methodology for designing robot controllers and simulators using enzymatic numerical P system models. SIMULATION p. 00375497241292347 (2024)
39. A. G. Florea and C. Buiu: PeP - an open-source software simulator of Numerical P systems and Numerical P systems with Enzymes 2017. [Online]. Available: https://github.com/andrei91ro/pep
40. Github simulation results folder https://github.com/radubobe/Mobile-Robots-Optimisation/tree/master/ICMC%20MC2%20Verification
41. MC2 User Manual, https://people.brunel.ac.uk/~csstdrg/courses/glasgow_courses/website_sysbiomres/software/mc2/UserManual.pdf
42. Valencia-Cabrera, L., Song, B.: Tissue P systems with promoter simulation with mecosim and p-lingua framework. Journal of Membrane Computing **2**(2), 95–107 (2020). https://doi.org/10.1007/s41965-020-00037-z
43. Verlan, S., Freund, R., Alhazov, A., Ivanov, S., Pan, L.: A formal framework for spiking neural P systems. Journal of Membrane Computing **2**(4), 355–368 (2020)
44. Younes, H.L., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. International Journal on Software Tools for Technology Transfer **8**(3), 216–228 (2006)
45. Yu, W., Wu, J., Chen, Y., Wu, Y.: Fuzzy tissue-like P systems with promoters and their application in power coordinated control of microgrid. Journal of Membrane Computing **5**(1), 1–11 (2023). https://doi.org/10.1007/s41965-022-00109-2
46. Zhang, G., Pérez-Jiménez, M.J., Gheorghe, M.: Real-life applications with membrane computing. Springer (2017). https://doi.org/10.1007/s00500-017-2906-y