applied sciences

MDPI

Article

Surrogate-Assisted Search-Based Testing of Robot Controllers Based on Enzymatic Numerical P Systems

Florentin Ipate * and Radu Traian Bobe

Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, 010014 Bucharest, Romania; radu.bobe@s.unibuc.ro
* Correspondence: florentin.ipate@unibuc.ro

Abstract

Surrogate-assisted search-based testing (SA-SBT) reduces simulation cost by replacing expensive runs with lightweight mathematical or data-driven approximations to efficiently guide metaheuristic search toward safety-critical system failures. While existing SA-SBT techniques predominantly employ black-box surrogates evaluated on short, static scenarios, they rarely exploit internal controller specifications or assess transferability to reactive closed-loop environments. In this paper, we aim to investigate how SA-SBT can be applied to a robot controller based on Enzymatic Numerical P systems (ENPS). We build and evaluate two surrogate strategies: a black-box classifier using road geometry and ENPS model identifier and a gray-box surrogate reconstructed directly from the controller internal rules. The classifier substantially reduces the number of required simulations while retaining nearly all true failures on independently generated test roads. In addition, we present and evaluate the gray-box surrogate, examining whether continuous fitness prediction, the mechanism underlying prior SA-SBT surrogates from the literature, transfers to a controller governed by a long, reactive control loop.

Keywords: robot controllers; simulation; surrogate modelling; membrane computing; enzymatic numerical P systems; search-based testing

1. Introduction

Over the past decades, the development of autonomous robotic technologies has attracted increasing research interest due to their growing potential in applications requiring reliable navigation, obstacle avoidance and intelligent operation in complex and dynamic environments [1,2]. Robot controllers represent essential elements of modern cyber-physical systems, providing computational and decision-making capabilities. In this regard, the main purpose of a robot controller is to translate high-level instructions into appropriate control commands that govern the motion and behaviour of robotic systems. Depending on the application and system architecture, the complexity of robot controllers may range from relatively simple rule-based implementations to highly complex frameworks incorporating real-time computation, sensor processing, optimisation and artificial intelligence techniques [3–5].

One of the most advanced categories of cyber-physical systems is represented by autonomous robots, which integrate information obtained from onboard sensors with internal models and algorithms to operate in dynamic environments. Their ability to function with limited human intervention makes autonomy particularly valuable in mobile robotics, industrial automation, exploration or service robotics. In such areas, robots must

Check for updates

Academic Editor: Yutaka Ishibashi

Received: 30 September 2026
Revised: 21 October 2026
Accepted: 26 October 2026
Published: 28 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

continuously interpret information from their surroundings, determine an appropriate next action and adapt their behaviour according to environmental conditions [6].

Obstacle avoidance controllers are widely used in autonomous robotic systems, particularly for navigation through unknown environments. They enable safe movement by continuously detecting potential obstacles and adjusting their trajectory to prevent collisions. However, the challenge lies not only in identifying obstacles, but also in responding to them efficiently while maintaining stable and effective motion. In this context, membrane computing [7] offers an alternative approach to implementing the computational processes required for robot control. In particular, *numerical P systems* [8,9], along with their extension, *enzymatic numerical P systems* [10,11], have demonstrated their suitability for robotic control applications, including autonomous navigation and obstacle avoidance.

Numerical P systems have shown strong potential for robotic control. Their rule-based structure facilitates real-time numerical processing, whilst the inherent parallelism allows for the simultaneous execution of multiple rules. These characteristics are particularly relevant to robotics, where information from multiple sensors may need to be processed simultaneously and converted into control actions under strict time constraints. Moreover, the enzymatic numerical P systems (ENPS) introduce an additional layer of control over rule execution by making the application of a production function dependent on the value of an associated numerical variable called *enzyme*. This mechanism allows multiple rules to be evaluated and executed in parallel within the same membrane, while providing greater control over the computational process. The conditional execution of rules has been used as a rigorous method of controlling autonomous robots with several inputs in real-time scenarios [12]. The computational power of enzymatic numerical P systems has been investigated in [13].

The use of ENPS model instances as the core component of an obstacle avoidance robot controller has been demonstrated in [14]. A test-driven methodology to model, simulate and test a robot controller based on several ENPS models has been proposed. Each model was constructed by collecting and addressing the limitations revealed through repeated simulation in Webots [15,16], over a set of road scenarios generated with a search-based testing tool, called AmbieGen [17,18]. While this approach successfully identified and resolved behavioural edge cases, the manual iterative process was computationally expensive, as each update demanded a complete execution of the full test suite. This limitation reflects a broader challenge studied in search-based software testing (SBST) [19]: the cost of repeatedly executing simulations to determine the outcome of individual test scenarios. Surrogate-assisted search-based testing (SA-SBT) [20,21] addresses this by training a surrogate model to predict scenario outcomes, relying on full simulations only to evaluate the most critical or uncertain candidates.

One of the most representative studies in this area was conducted by Ben Abdessalem et al. in 2016 [22], who combined multi-objective search with a neural network surrogate to test a pedestrian-detection ADAS, reporting substantial reductions in simulation calls without loss of test effectiveness. The proposed approach was surveyed in 2023 by Nejati et al. across sixteen key studies that utilise surrogate modelling to scale search-based system-level testing of compute-intensive autonomous systems [23]. As the authors of the study noted, the gray-box technologies remain comparatively under-explored in this context, despite offering an effective compromise when partial structural knowledge of the system is available.

In this paper, we study how SA-SBT can reduce the simulation cost of testing robot controllers based on ENPS. The test-driven design of ENPS-based robot controllers originates in our previous work, where we developed and manually tested several ENPS model instances [12,14]. As a new contribution, we present two surrogate strategies differentiated by the degree of structural knowledge they exploit: a black-box classifier relying on road ge-

ometry and a gray-box surrogate built directly from the controller symbolic kinematics and sensor layout. We test the classifier on an independent set of road test scenarios imported in Webots and measure how many simulations it saves. We test the gray-box surrogate against the robot trajectories to determine whether continuous fitness prediction (the approach used in prior SA-SBT work) transfers to our controller instances.

We formulate the investigation around three research questions:

RQ1. Can a surrogate model reduce the cost of the simulation-based testing campaign for an ENPS-based controller, while preserving most of its fault-detection capability?

RQ2. To what extent can a surrogate model be constructed by exploiting the formal specification of an ENPS controller?

RQ3. How effectively does continuous fitness prediction transfer to long, reactive control loops?

The paper is structured as follows. Section 2 states the essential preliminaries. Section 3 presents the ENPS models and the underlying controller logic. Section 4 describes the methodology and setup. Sections 5 and 6 introduce the black-box classifier (RQ1) and the gray-box surrogate (RQ2). Section 7 evaluates the closed-loop system (RQ3) and summarises the main results of this study. Finally, Section 8 states the conclusions.

2. Preliminaries

This section presents the foundational concepts underlying our approach. First, we introduce the enzymatic numerical P systems (ENPS), which serve as the core computational framework of the System Under Test (SUT). Then, we introduce Search-Based Software Testing (SBST) and review the SA-SBT taxonomy proposed in the recent literature.

2.1. Enzymatic Numerical P Systems

The core component of the robot controller under test is an Enzymatic Numerical P system (ENPS) model. As already stated, multiple instances of the model were previously developed in an incremental manner, each addressing the limitations revealed by testing the previous one. In the current SUT, the models compute the two wheel velocities, taking as input proximity sensor readings associated with static weight values.

Enzymatic Numerical P systems [24,25] represent a class of membrane computing models that inherit only their membrane structure while processing numerical variables inside each compartment. The variables are initialised prior to computation and evolve in accordance with predefined evolution rules (programs). During each computation step, active rules from multiple membranes execute concurrently, enabling parallel and distributed updates throughout the system.

An ENPS is an extension of the standard numerical P systems (NPS). In contrast with NPS, ENPS incorporate enzymes as internal control elements within the programs. An enzyme represents a numerical variable whose value dictates the activation of a rule according to the following principle: a program is triggered only when its assigned enzyme value strictly exceeds the minimum value among the variables involved in the respective production function. This mechanism provides enhanced precision over numerical updates, making ENPS well-suited for controlling robotic systems [12,26].

The enzymatic numerical P system is defined by the following tuple:

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

where:

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

- μ is the membrane structure;
- Var_i is a set of variables from membrane $i, 1 \leq i \leq m$;
- $Var_i(0)$ are the initial values of the variables from region $i, 1 \leq i \leq m$;
- 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 one of the two following forms:

  (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. For each specific membrane i, variables $v_1, v_2 \ldots v_{m_i}$ are located within the membrane containing the programs, and to its parent and child compartments.

  (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, $e_i \notin \{x_{1,i}, \ldots, x_{k,i}, v_1, \ldots, v_{m_i}\}$. Inspired by the biological enzyme, the value of the enzymatic variable e_i dictates whether the program can be applied at time t. More precisely, the program will be selected for execution only if

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

All the selected rules are executed simultaneously in their membranes. When a program $Pr_{l_i,i}$, $1 \leq l_i \leq m_i$ is applied by the system at time $t \geq 0$, the resulting value

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

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

At time step t, the values of all variables participating in the production functions of the selected programs at the previous step are reset to zero (consumed). The new value becomes the sum of the proportions assigned to it by the repartition protocols, or remains zero if it is not targeted by any protocol.

2.2. Surrogate-Assisted Search-Based Testing

Search-Based Software Testing (SBST) is an automated testing paradigm that reformulates the generation and optimisation of test cases as a metaheuristic search problem, leveraging techniques such as genetic algorithms or local search to efficiently navigate vast input spaces [27,28]. Rather than relying on manual test creation or random exploration, SBST uses a domain-specific fitness function to guide optimisation algorithms toward automatically discovering inputs that maximise code coverage, expose boundary conditions, or uncover critical faults within the system under test (SUT).

To scale search-based testing to compute-intensive systems, Surrogate-Assisted Search-Based Testing (SA-SBT) integrates metaheuristic search with machine learning surrogate models (such as neural networks, decision trees, or regression models) [22]. In general, surrogates improve computational efficiency in two distinct ways: first, by identifying critical regions within the input space to guide test generation toward high-risk scenarios and second, by estimating fitness values or execution outcomes instead of running expensive physical executions. According to the taxonomy proposed by Nejati et al. [23], SA-SBT research can be categorised along several dimensions:

- **Purpose**. Refers to how surrogates are integrated into the search process and which search elements they aim to improve;
- **Usage**. Defines how surrogate predictions govern test generation (e.g., Single Surrogate Test Generation, Single Surrogate Test Filtering, Ensemble Surrogate Generation);
- **Subject**. Represents the artifact approximated by the surrogate model;
- **Design**. Indicates how the surrogate is constructed (manually or automatically via data-driven dynamic learning loops [29]);
- **Type**. Captures the core properties of the surrogate across two dimensions: *model type* (black-box, gray-box and white-box) and *output type*. Depending on the target task, surrogates generate either point estimates (e.g., individual fitness scores) or continuous time-series data reflecting system state evolution;
- **Context**. There are three main usage contexts for surrogates: *Model-in-the-Loop (MiL)*, *Software-in-the-Loop (SiL)* and *Hardware-in-the-Loop (HiL)* [30,31];
- **Scope**. Specifies the extent of the SUT search domain covered by the surrogate model (i.e., global view and local view of the search space);
- **Evaluation Metrics**. Refers to the metrics used to evaluate a SBT method (e.g., number of iterations, execution time).

Among these dimensions, the *model type* determines what the surrogate knows about the system it approximates. Our SUT has a hybrid observability profile: the ENPS system, which computes wheel velocities from sensor readings, is fully explicit, symbolic and machine-readable, whereas the physical interactions that are governed by the ENPS remain only partially known, as the virtual environment provided by Webots is governed by equations whose parameters (e.g., wheel slip) are not fully accessible. Consequently, there is a contrast between the white-box nature of the ENPS component and the gray-box nature of the physical environment the controller operates in. This structural setup motivates the surrogate strategies that are introduced and evaluated in this paper.

3. ENPS-Based Robot Controller

The robot controller is deployed on E-puck, a small differential-drive educational robot. It has a diameter of 7.4 cm and is simulated in Webots using a dedicated PROTO object [32]. The robot is equipped with eight proximity sensors that are placed radially around the robot perimeter. Our experiments revealed that for the obstacle avoidance scenarios considered in our approach, the two rear-facing sensors are never activated. Thus, in our experiments, we used the readings from the rest of six proximity sensors: a front pair placed at 3 cm in front of the centre and 1 cm either side of the axis, two front-lateral sensors at 2.2 cm in front and 2.5 cm to the side and two lateral sensors at 3.1 cm to the side, all of them mounted on the circular body. Within the Webots simulation environment, the intensity of the reflected infrared light is converted into proximity values using a lookup table calibrated on the real robot [33]. The sensors have a maximum functional range of 4 cm, beyond which they return a value of zero (meaning that no obstacle is detected). Within this threshold, they report proximity values calculated by the Webots internal sensor model, which remain highly sensitive to the color, texture and material of surrounding obstacles. E-puck has two wheels, each with a radius of 2.05 cm and driven independently by a dedicated motor, giving the controller direct control over left and right wheel angular velocity at every simulation timestep. The controller sends angular velocities (rad/s) to each wheel at every simulation step of 32 ms, up to a maximum of 6.28 (rad/s) [34,35]. Figure 1 illustrates the E-puck robot within the simulation environment.

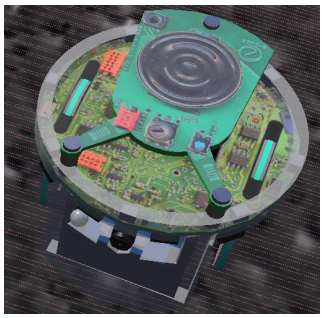


Figure 1. Simulated E-puck robot captured in Webots interface.

3.1. Enzymatic Numerical P System Models

The obstacle avoidance robot controller present in the current work is based on ENPS. Building on the design presented by Buiu and Florea [36], in our previous work we adopted a test-driven methodology to construct multiple instances of ENPS models that can be seamlessly integrated as the core component of the robot controller [14]. Each model was developed after analysing the behaviour of the previous one during manual tests performed using E-puck in Webots on several curved roads. The baseline model described by Buiu and Florea is referred as Π_{M_1}.

Each ENPS model takes as input proximity sensor values and their associated weight values (numerical coefficients that are multiplied by sensor readings). Consequently, assigning opposing weight distributions can lead to differential steering, enhancing the stability and natural movement of the robot. By taking these sets of sensor values and weights at each simulation step, the ENPS model has several programs that are responsible for continuously generating the left and right wheel speeds.

3.1.1. Basic Model with Rotation

The *Basic Model with Rotation* Π_{M_2} was developed after analysing the limitations of the baseline model [37]. As already stated, each proximity sensor $prox_i, 1 \leq i \leq n$ has associated weight values, $weightLeft_i, weightRight_i, 1 \leq i \leq n$. The model has dedicated compartments that compute the aggregated *weightLeft* and *weightRight*, as follows:

$$weightLeft = \sum_{i=1}^{n} weightLeft_i \cdot prox_i \tag{6}$$

$$weightRight = \sum_{i=1}^{n} weightRight_i \cdot prox_i \tag{7}$$

Before proceeding with the formulas for computing the wheel speeds, the following function has to be defined:

$$f(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

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

$$leftSpeed = cruiseSpeed \cdot weightLeft + f(weightLeft) \cdot cruiseSpeed \tag{9}$$

$$rightSpeed = cruiseSpeed \cdot weightRight + f(weightRight) \cdot cruiseSpeed \tag{10}$$

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

The above equations are represented as programs inside the membranes of Π_{M_2}. The mathematical representation of the resulting model is the following:

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0))) \tag{11}$$

where:

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

  The meaning of each variable from the model is detailed below:

- x_{s_l} and x_{s_r}, residing within membrane s, denote *leftSpeed* and *rightSpeed*; membrane s accumulates the sum of products once the rules associated with membranes c_i ($1 \le i \le k$) have been executed;
- x_{w_l} and x_{w_r}, located in membrane w, designate *weightLeft* and *weightRight*, computed according to Equations (6) and (7);
- x_{s_c}, from s_c, corresponds to *cruiseSpeed*;
- variables x_{c_i,s_l} and x_{c_i,s_r} of membrane c_i express the sensor readings scaled within membranes s_i, whereas x_{c_i,w_l} and x_{c_i,w_r} contain the values of $weightLeft_i$ and $weightRight_i$ computed within membranes w_i;
- each weight tuple, ($weightLeft_i$, $weightRight_i$), is represented as (x_{w_i,w_l}, x_{w_i,w_r}) and resides in membranes w_i for $1 \le i \le k$;
- each proximity sensor $prox_i$ is linked to a dedicated membrane s_i containing a single variable, $x_{s_i,i}$ ($1 \le i \le k$), which stores the most recent measurement reported by the sensor.

As stated earlier, the ENPS model computes the *leftSpeed* and *rightSpeed* at every simulation step. According to the operational logic of ENPS that was presented in the previous section, if a variable is not implied in any program at one step, its value is accumulated. As the wheel speeds have to be computed according to the latest sensor readings, x_{s_l} and x_{s_r} are multiplied by zero in the production function from membrane s and the previous values are consumed.

The *weight* membrane w contains two dedicated rules that aggregate weight contributions and apply the *cruiseSpeed*. Within this membrane, the accumulated values of *weightLeft* and *weightRight* are multiplied by *cruiseSpeed*, with a fallback mechanism (adds *cruiseSpeed* if weight is 0), as formulated in (9) and (10), to compute the final wheel velocities using enzymes. Each compute membrane $c_i, 1 \le i \le k$ contains two programs that compute the weighted contribution of each proximity sensor. Furthermore, membranes s_i propagate duplicate values to the corresponding left and right computation variables (x_{c_i,s_l}, x_{c_i,s_r}) located in the parent compartment. Similarly, membranes w_i duplicate each pair of weights, $weightLeft_i$, $weightRight_i$, providing copies (x_{c_i,w_l}, x_{c_i,w_r}) to the corresponding membrane c_i.

3.1.2. Refined Model

Although the previous model demonstrated good results in passing the obstacle avoidance scenarios, a new "Refined Model", Π_{M_3}, has been developed to enhance the natural movement of the robot, avoiding "zig-zag" motions between the road borders. By using this model as the core component of the controller, the robot recentres after avoiding an obstacle. This new feature was achieved by integrating a new membrane, called "Direction", into the control logic of the previous model. Within the "Direction" membrane, a variable called "state" is used by the programs, replicating a finite state machine with the following states:

(a). state 0—straight line movement (no obstacle reported)
(b). state 1—the robot is avoiding an obstacle (at least one sensor reported a positive value)
(c). state 2—the robot keeps advancing for a fixed time before recentering, after obstacle avoidance
(d). state 3—the robot turns back towards the lane axis

It can be noticed that there is a link between the value of the variable *state* and the actual action of the robot. Through this mechanism, the variable conditions the production rules that compute the left and right wheel speeds according to the operational rules that were previously described for Π_{M_2}. Even though the integration of this mechanism increases the total number of variables and programs, manual tests have demonstrated that the robot follows noticeably more natural trajectories.

Figure 2 illustrates how the *state* variable evolves during an obstacle avoidance manoeuvre. Whenever an obstacle is detected, the controller moves to state 1. Once the obstacle is no longer sensed, the controller goes through the two timed phases described above (state 2 and state 3) and then returns to state 0. As marked by the dashed arrows, if a new obstacle is detected during these phases, the manoeuvre is interrupted and the controller returns to state 1. For the comprehensive formal definition of Π_{M_3}, the reader is invited to consult [12].

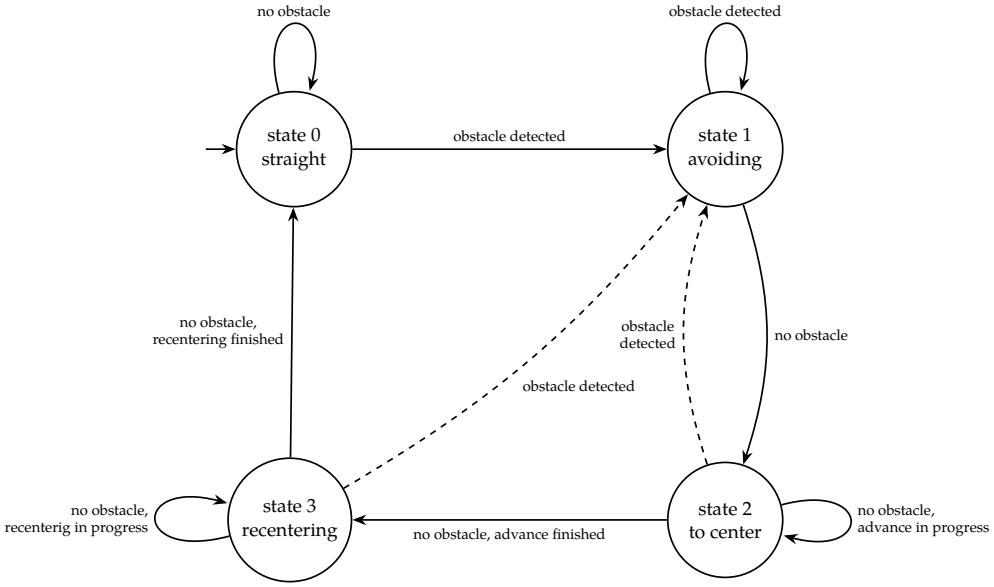


Figure 2. State transition diagram of the "Direction" membrane of Π_{M_3}.

3.2. Controller Architecture

The above models are designed to provide the wheel speeds of the robot, according to the input collected from the simulation environment. Each of the above mentioned model instances (Π_{M_1}, Π_{M_2}, Π_{M_3}) can be seamlessly integrated in the controller logic. The robot

controller mediates real-time information exchange between Webots and the selected ENPS instance. Consequently, at each simulation step, the sensor readings are retrieved from Webots and fed as input data into the ENPS model, together with static weight parameters. Then, the model computes the wheel speeds according to the logic described above that is encoded within its membrane hierarchy. The returned values are assigned to the left and right motors to drive the robot through the obstacle avoidance scenario. This operational cycle of the proposed ENPS-based controller is formalised as pseudocode in Algorithm 1, as defined in [37].

Algorithm 1 Main loop of the robot controller

```
repeat
    for i=1 to number_of_sensors do
        sensor_membrane(i) ← value(i)

    run one simulation step
    read leftSpeed, rightSpeed from ENPS
    leftMotor ← leftSpeed
    rightMotor ← rightSpeed
until the end of the road or E-puck goes out of the road
```

The robot controller was implemented as a Python script and integrated into the Webots user interface. It can be observed that it manages the interaction between the robot and the ENPS model. The ENPS simulation is realised via PeP, an open-source tool that facilitates the execution of (Enzymatic) NPS [38]. To perform the computation of wheel speeds, the ENPS model definition (including membrane structure, compartments, variables with initial values and programs) is formalised into a *.pep* file. PeP can be executed directly from the command line or called programmatically (as implemented in the controller logic). During simulation, the Python controller invokes one execution step of the ENPS model with the input values, retrieves the resulting wheel velocities and assigns them to the corresponding Webots motors.

Beyond its core function of executing numerical P systems, PeP additionally generates a complete trace of the execution, containing all the variable values during each simulation step. These traces record every sensor reading, weight value and motor speed. They allowed us to study the controller behaviour during the testing phase in certain circumstances and to reconstruct the robot kinematics independently of Webots, as detailed later in this paper.

4. Methodology and Setup

This section describes the infrastructure built to obtain data about the controller behaviour across a range of road test scenarios.

A notable distinction between the current study and existing SA-SBT methods for CPS is that we do not need to construct a new surrogate model to generate road tests. From the initial manual testing phase, we integrated AmbieGen, an SBST tool that natively employs surrogate functions to guide the evolutionary process [17,39]. AmbieGen is a framework used for autonomous systems testing, providing road test scenarios as *.json* files. The tool was originally applied to test case generation for a vehicle LKAS system. By varying AmbieGen configuration parameters (e.g., minimum road length, maximum straight segment length), we generated a diverse set of roads. A custom Python script was then used to extract the centreline coordinates from the generated output and export them as ordered point sequences into individual text files. Finally, these spine points were used to reconstruct the roads with the specified target widths inside the Webots environment. Given that AmbieGen served a similar purpose and its effectiveness has already been

demonstrated [40], we were able to utilise its generated roads within Webots with only minimal scripting adjustments.

Initial evaluation of the ENPS-based controller primarily relied on human inspection, consisting of manual execution and monitoring to check whether the robot completed the road. As expected, this method does not scale to the hundreds of simulations required for a surrogate-assisted campaign. Moreover, it does not provide quantitative metrics to show how close the robot was to completing the task. Accordingly, one of the priorities was to replace the human inspection with an automated oracle. The oracle projects the robot position on the road spine at every simulation step and computes the distance travelled along the road and the distance from the centreline, allowing determination of out-of-bounds conditions.

The obtained oracle has three possible outcomes:

- **SUCCESS**, if the robot passes the test by reaching the end of the road;
- **OOB (out-of-bounds)**, if the lateral distance exceeds half the road width minus the robot body radius;
- **STUCK**, if the robot fails to advance for 90 s (indicating that it is spinning without progress).

After analysing several scenarios, we determined that a separate FAILURE outcome is not necessary, since the three conditions above are sufficient to characterise every outcome we observed.

Besides the success/failure (due to out-of-bounds or stuck), it is useful to record information about the robot behaviour during the test. By using the position trace, we derived three continuous fitness functions that are computed after each simulation:

- f_1 *(margin to failure).* As mentioned, a binary outcome hides details about controller performance, treating a robot that fails during the last computation steps the same as one that is not capable of avoiding any obstacle. To address this, we introduced f_1, defined as the minimum distance remained until an eventual OOB. A positive value indicates the smallest margin observed during a successful run, whilst $f_1 < 0$ indicates that the robot went OOB.
- f_2 *(path progress).* This function measures how far the robot managed to advance relative to the total road length. The function is useful to determine how much of the obstacle avoidance task was completed during an unsuccessful test.
- f_3 *(smoothness).* While f_1 and f_2 represent relevant spatial metrics, they do not provide insights into the specifics of the robot navigation. To quantify this behaviour, we defined f_3 as the mean absolute change in left and right wheel velocity between consecutive simulation steps. Thus, a robot that oscillates between sharp turns produces a higher value, whilst smooth movement results in lower values.

In line with prior work on surrogate-assisted testing [22], these three functions provide continuous evaluation metrics. Here, however, we use them mainly for diagnostics and dataset construction rather than direct regression targets, as it will be detailed in the next sections.

Test Campaign

The road test scenarios employed in previous work [9,37] use a fixed road width of 0.2 m. The model evaluation results showed that Π_{M_2} and Π_{M_3} pass nearly every generated scenario. This difference between the results of Π_{M_1}, which fails in the majority of cases and the subsequent models can be explained by the structure of the models. More precisely, Π_{M_1} adds the weighted sensor contributions into the wheel speed. As a consequence, the robot never stops advancing. Instead, Π_{M_2} and Π_{M_3} compute wheel speed multiplicatively,

as denoted by 9 and 10. Consequently, the sum of the wheel speeds can be zero, allowing the robot to stop and rotate in place. Our analysis conducted on the model traces showed that this important distinction accounts for the high failure rate of Π_{M_1}, in comparison to enhanced models. However, in the context of the current initiative, the near-total success rate of Π_{M_2} and Π_{M_3} may represent a "disadvantage", as there is little failure variation to learn from. We therefore made road width a configurable parameter in the Webots virtual interface. We tested widths down to 0.10 m and observed that below approximately 0.12 m, the resulting configuration is unstable: the physics engine repeatedly reports the warning message *"physics step could not be computed correctly"* and, in some cases, removes the robot from the scene. We therefore consider the following road widths: 0.12, 0.15, 0.20 and 0.25 m. All selected widths remain within the stable range of the simulator, thereby following Nejati et al. [23] in avoiding "failures caused by invalid inputs and not due to system faults".

To collect data across the resulting combination of roads and models used, we built a runner that generates the Webots world file for each specific road, configures the controller with the targeted ENPS model and launches the simulator in headless mode. Once the simulation is completed and the oracle results are delivered, the simulation process is terminated. Consequently, we obtained a *training set* of **100 roads** and a separate *validation set* of **50 roads**, each being simulated at all four widths and with the controller using Π_{M_1}, Π_{M_2} and Π_{M_3} models, yielding 1200 and 600 simulation runs respectively. Table 1 summarises the characteristics of the two road sets, in terms of length, curvature, turning and observed failure rates. Across all 1800 simulations, successful runs lasted an average of 285 simulated seconds, whereas out-of-bounds (OOB) failures lasted 60 s and STUCK failures averaged 239 s. Figure 3 illustrates three different road tests from the training set.



(**a**) (**b**) (**c**)

Figure 3. E-puck robot navigating different test road topologies in Webots.

Table 1. Summary of the training and validation road sets. The minimum curvature radius characterises the tightest bend of a road, while total turning captures the absolute change in heading along the entire road. Failure rates are computed over all runs (12 per road); geometric values are mean ± standard deviation.

Characteristic	Training	Validation
Road length (m)	3.26 ± 1.47	2.67 ± 0.61
Minimum curvature radius (m)	0.23 ± 0.11	0.20 ± 0.06
Total turning (rad)	5.59 ± 1.54	6.63 ± 2.10
Failure rate, all runs	39.9%	37.7%
Π_{M_1}	96.0%	97.5%
Π_{M_2}	10.5%	7.5%
Π_{M_3}	13.2%	8.0%

The road testing campaign was executed in Webots R2025a, while the controller was implemented in Python 3.11. The entire development and evaluation process was hosted on a Windows 10 machine equipped with an Intel Core i7-8565U CPU, Intel UHD Graphics 620 (128 MB) and 16 GB of RAM.

5. Test Outcome Prediction

We first investigate whether a surrogate model that has access only to road geometry and ENPS model label (not the controller internal rules) can accurately predict test outcomes. To some extent, this approach aligns with the black-box surrogate strategies established by Nejati et al. taxonomy [23]. Moreover, it provides a baseline for the gray-box surrogate that is built directly from the ENPS mechanism, as detailed in the next section.

Each test road is initially defined by the road spine extracted from AmbieGen. By using the road spine, we extracted a fixed-length geometric descriptor, comprising several aggregate data about the road curvature profile, such as total length, maximum curvature, mean absolute curvature or net turning angle. Then, we added the road width and the model used (Π_{M_1}, Π_{M_2} or Π_{M_3}) and obtained a vector encoding road, width and model information.

Given the feature representation, the next critical decision that required careful consideration was selecting a classifier family capable of exploiting it effectively. Table 2 presents the comparison of five candidates using 5-fold cross-validation grouped by road (20 roads per fold, 0.5 decision threshold): logistic regression, two neural networks (multilayer perceptrons with hidden layers of (64, 32) and (100, 100)), random forest and gradient boosting.

Table 2. Comparison of different surrogate model families.

Model	Accuracy	Precision	Recall	AUC
Per Model Majority Baseline	90.8%	96%	80.2%	-
Logistic Regression	89.5%	92.3%	80.4%	0.930
MLP(64, 32)	88.3%	85.2%	85.6%	0.916
MLP(100, 100)	87.3%	82.9%	86.0%	0.916
Random Forest	92.7%	95.4%	85.8%	0.930
Gradient Boosting	91.5%	92.9%	85.2%	0.931

By analysing the results presented in Table 2, one can notice that all non-linear methods achieved comparable discrimination (AUC 0.916–0.931). The "Per Model Majority Baseline" predicts the most frequent outcome for each ENPS model, without using any road information. We select Random Forest for its superior accuracy among all models (92.7%) and the highest precision among the learned methods (95.4%). As our proposed methodology involves that any predicted failure must be confirmed by running the test case in Webots, achieving higher precision reduces unnecessary simulations caused by false positives.

The Random Forest uses 400 trees with a minimum of 2 samples per leaf; all other hyperparameters take the scikit-learn (v1.9.1) defaults (Gini criterion, unlimited depth, bootstrap sampling, no class weights) and the random seed is fixed to 0. No hyperparameter search was performed. We trained the selected random forest on the full training set, consisting of 100 roads and 1200 runs (100 roads × 4 widths × 3 ENPS models) and then evaluated it on the validation set (50 roads, 600 runs). We recall that in order to obtain an unbiased estimate, the validation set was generated independently and contains no roads from the training set. Under these conditions, the surrogate model achieved the following performance on the validation set:

- *Accuracy: 91.2%*;
- *Precision: 89.1%*;
- *Recall: 87.2%*;
- *AUC: 0.971*.

Of 226 true failures in the validation set, 197 are correctly flagged, 29 are missed, and 24 successful runs are incorrectly flagged as failures.

Figure 4 shows the ROC curve of the classifier on the validation set. Since the failures of the baseline model Π_{M_1} are mostly identifiable from the controller label, the curve restricted to the improved models Π_{M_2} and Π_{M_3} is also reported.

Although the above mentioned metrics represent an accurate way to evaluate the surrogate, its practical applicability is to filter which scenarios are submitted to simulation in Webots. Our solution does not merely return a binary pass/fail outcome. Instead, it produces a continuous probability that a given scenario will fail. Consequently, we define six decision thresholds that determine the probability value above which a scenario should be labeled as a predicted failure. The probability threshold is an operating parameter and does not affect the ranking quality of the classifier (summarised by the AUC). For each of these thresholds, Table 3 reports the fraction of the 600 validation scenarios that would be simulated and the fraction of the 226 true failures that were present in the validation set.

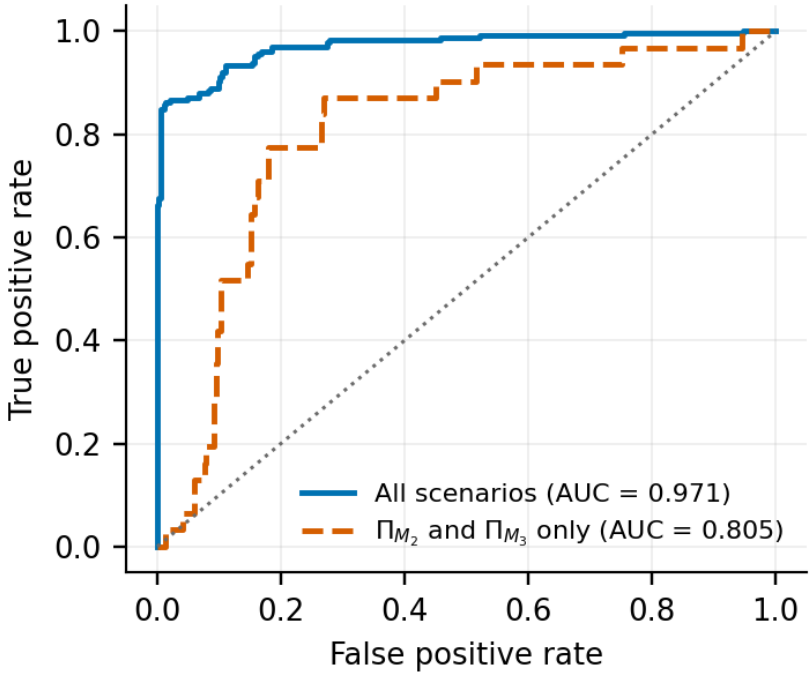


Figure 4. ROC curves on the validation set. The solid line shows the performance across all scenarios; the dashed line restricts evaluation to Π_{M_2} and Π_{M_3} scenarios only. The dotted diagonal represents a random classifier baseline.

It can be observed that lowering the threshold recovers a larger share of true failures, at the cost of running more simulations. For example, at a threshold of 0.20, 306 simulations are required (51% of the total, 49% avoided) while retaining 97% of true failures (7 missed). Increasing the threshold to 0.30 lowers the number of simulations run (293, representing 49% of the total), with the same 7 failures missed, while a threshold of 0.50 reduces simulations to 37% but at a considerably higher cost, missing 29 failures (13%). In terms of time, the saving is more significant than the drop in simulations run. More precisely, at threshold 0.30, a reduction of 71% of simulated time is observed, since the discarded scenarios are mostly long successful runs rather than short failures. We therefore chose a threshold of 0.30

as it achieves the same failure detection count as 0.20 while requiring 13 fewer simulations and less total simulation time. The threshold was selected on the validation roads and the random forest retained its configuration, as defined earlier.

Table 3. Simulation savings at different probability thresholds.

Threshold	Simulations Run	Simulated Time Run	Failures Found
0.05	469/600 (78%)	69%	224/226 (99%)
0.10	395/600 (66%)	52%	223/226 (99%)
0.15	330/600 (55%)	38%	222/226 (98%)
0.20	306/600 (51%)	33%	219/226 (97%)
0.30	293/600 (49%)	29%	219/226 (97%)
0.50	221/600 (37%)	12%	197/226 (87%)

Figure 5 illustrates the cost-detection trade-off on the validation set. Scenarios are run in decreasing order of their predicted failure probability. The curve shows the proportion of total simulated time against the share of true failures found. The marker shows the adopted threshold of 0.30. It can be observed that the classifier concentrates most failures among the first scenarios it ranks, as at the selected threshold, 29% of the total simulated time is spent and 97% of the failures are found.

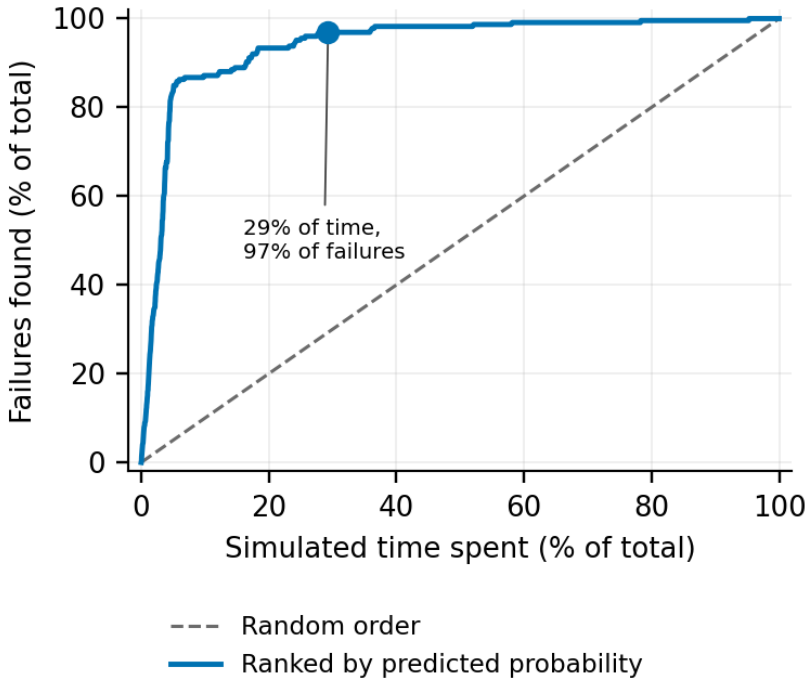


Figure 5. Cost–detection trade-off on the 600 validation scenarios.

6. Surrogate from ENPS Specification

In the previous section, we introduced and evaluated a surrogate that has no access to the controller internal rules. However, unlike prior studies from the SA-SBT literature, the black-box approach was chosen deliberately, as the ENPS-based controller rules are fully available and made explicit by the ENPS formalism introduced in Section 2. In this section, we build upon this availability and examine whether the exact controller behaviour

during the simulations can be reconstructed entirely from its symbolic specification, without performing any simulation in Webots.

As described in Section 3.1, the two motor velocities (*leftSpeed*, *rightSpeed*) are computed by ENPS models. Their values at each simulation step can be exported from PeP. Therefore, the robot trajectory can be reconstructed from the traces by using the following differential-drive kinematics:

$$v = \frac{(leftSpeed + rightSpeed)}{2} \cdot r \cdot k_v \tag{12}$$

$$\omega = \frac{(rightSpeed - leftSpeed)}{L} \cdot r \cdot k_\omega, \tag{13}$$

where r = 0.0205 m represents the wheel radius and L = 0.052 m is the axle length. The scaling factors k_v and k_ω denote the wheel slip that is not captured by the kinematic model and are calibrated by minimising position error against ground-truth GPS trajectories.

We obtained the values of k_v and k_ω by independently fitting the kinematic model to several simulation traces. The obtained values are consistent, with coefficients of variation below 4%. Consequently, it can be affirmed that the wheel slip represents a stable characteristic of the simulated robot rather than an artefact of curve-fitting on individual trajectories. Applying the averaged parameters ($k_v = 0.977$, $k_\omega = 0.946$) across the runs has led to a reconstruction of trajectories with 4.1–11.1 mm mean error over tests with up to 8000 simulation steps. Table 4 presents the results for four road tests from the set with different geometries:

Table 4. Kinematic reconstruction accuracy on different road tests.

Test	Steps	Mean Position Error
Road 1 (width = 0.15)	6604	11.1 mm
Road 5 (width = 0.15)	6143	4.7 mm
Road 10 (width = 0.12)	4734	4.1 mm
Road 15 (width = 0.20)	8007	8.2 mm

Figures 6 and 7 provide a visual representation of the trajectory reconstructed from the ENPS controller onto the simulator ground truth. We refer to this reconstruction process as *kinematic reconstruction* throughout the remainder of the paper. They also plot the position error at each simulation step for Road 1 and Road 10. These two roads correspond to the minimum and maximum mean position errors reported in Table 4. We chose to include the road spine as a geometric reference rather than the full road widths. This aligns with the testing oracle, which measures the lateral deviation of the robot centre from the spine when detecting OOB, as described in Section 4. The above results indicate that having the wheel velocities from the ENPS traces can lead to accurate trajectory reconstruction. However, as the objective is to bypass Webots simulations, the reverse direction is also necessary. Accordingly, we aimed to reconstruct the sensor readings during the simulation and give them as inputs for the ENPS in order to return the estimated wheel speeds without access to Webots.

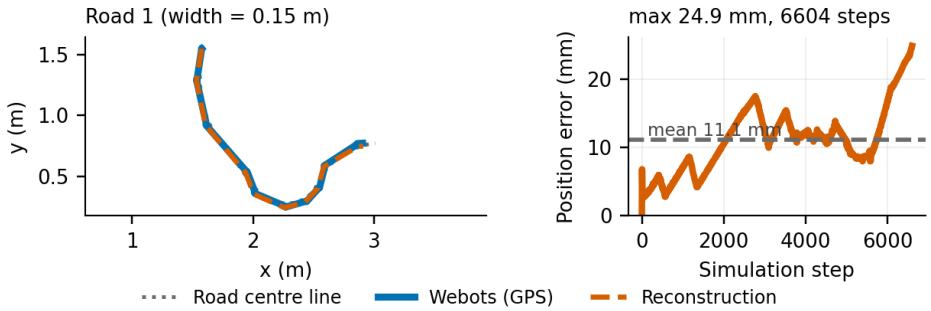


Figure 6. Trajectory reconstruction and mean position error for Road1.

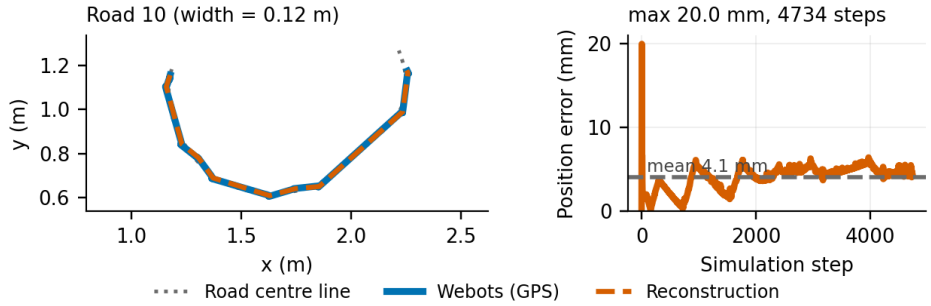


Figure 7. Trajectory reconstruction and mean position error for Road10.

According to Webots Documentation [41], the formula used by the simulator to compute the reflection factor f for infrared rays is the following:

$$f = 0.2 + 0.8 \cdot red_level \cdot (1 - 0.5 \cdot roughness) \cdot (1 - 0.5 \cdot occlusion) \quad (14)$$

We calibrated f empirically by using a static E-puck instance at known distances from a flat surface, varying the distance and orientation.

As the reported distance is scaled by f, we incorporated this factor into the geometric model by using the true sensor mounting position that was found in the *E-puck.proto* specification and evaluated against ground-truth GPS positions.

Both kinematic reconstruction and sensor model were developed against simulation data. However, for the proposed surrogate, at each computation step, the above sensor model should be evaluated at the reconstructed position exemplified in Table 4. We evaluated the kinematic reconstruction and sensor model on 10 roads from the validation set. The roads were selected to cover the four width values and have geometries of varying complexity, by following the metrics defined in Table 1. For each simulation, Figure 8 plots the mean position error of the reconstructed trajectories (by following the above presented method) against the sensor match rate computed at the reconstructed position.

On the considered representative set, the mean position error ranges from 4.1 mm to 19.3 mm (median 9.8 mm) and the sensor match at the reconstructed position ranges from 1% to 39.2% (median 10.1%). In comparison, with the ground-truth position, the sensor model reproduces 64–92% of the active readings (median 79.7%). Thus, the reconstructed position match is considerably lower than the ground-truth match across all the roads (median gap 69.6%). To further analyse the relationship between these metrics, a detailed examination for the same roads for which the mean position error was reported in Table 4 is presented. Table 5 reports the accuracy under both conditions: reconstructed position and GPS ground-truth.

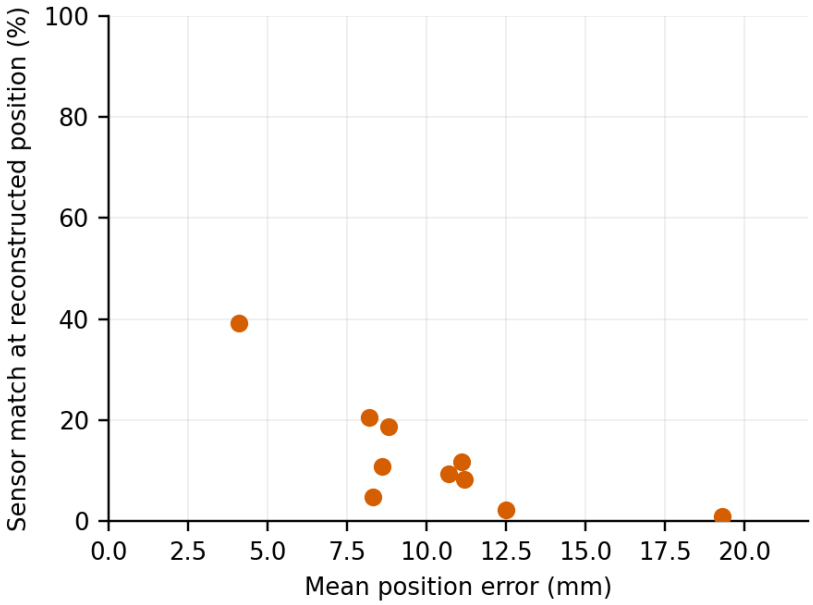


Figure 8. Sensor match for 10 roads. The filled circles represent the reconstructed position against the mean position error of the reconstruction.

By analysing the values of the metrics presented in Tables 4 and 5, it can be noticed that higher position error corresponds to lower sensor match. For example, Road 1 (width = 0.15) has the biggest mean position error (11.1 mm) and the lowest sensor match on reconstructed position (11.8%). In contrast, Road 10 (width = 0.12) has the lowest mean position error (4.1 mm) and the highest sensor match (39.2%). This can be explained by the fact that a sensor misreading alters the subsequent wheel velocities that are computed by ENPS. In the intended surrogate, the reconstructed sensor readings are fed back to the ENPS controller, so a sensor deviation changes the wheel commands, which in turn affect the subsequent robot positions.

Table 5. Sensor model accuracy: ground-truth vs reconstruction.

Test	Sensor Match (Ground Truth)	Sensor Match (Reconstructed Position)
Road 1 (width=0.15)	76.6%	11.8%
Road 5 (width=0.15)	80.3%	27.5%
Road 10 (width=0.12)	79.1%	39.2%
Road 15 (width=0.20)	74.5%	20.6%

Figure 9 shows that the sensor discrepancies translate into wheel command deviations that accumulate along the simulation. However, the results do not indicate a failure of the reconstruction, but they raise a question regarding the sensitivity of the composed system. This perspective is discussed in the next section.

As the evaluation was conducted in a simulation environment, it should be noted that a real-world deployment may introduce additional sources of inaccuracies that are not captured by the kinematic model. Consequently, the generalisability of the results on other simulators or physical hardware remains to be validated.

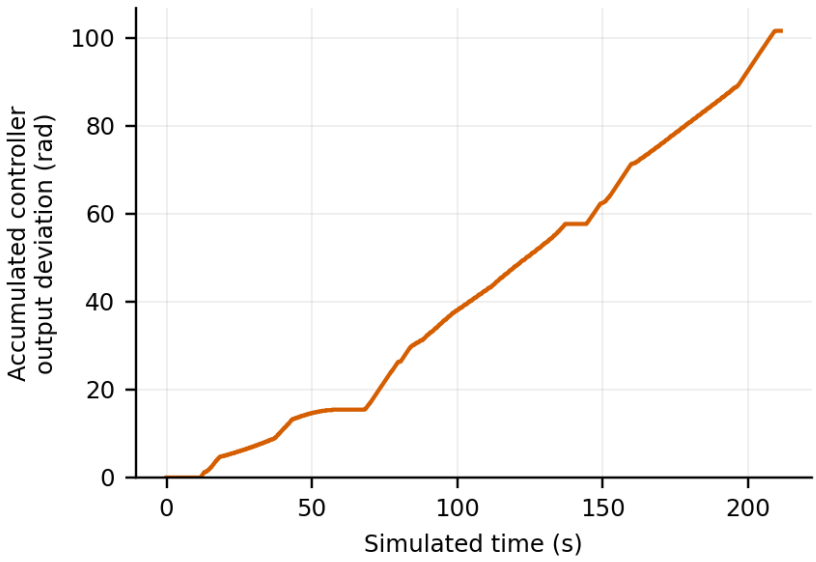


Figure 9. Accumulated controller-output deviation on Road 1 (width = 0.15 m).

7. Discussion

Section 6 showed that even though the kinematic and sensor models perform well separately, their aggregation sharply degrades sensor accuracy. The reflectance factor f introduced in the previous section has a direct geometric consequence that can contribute to the explanation. More specifically, with $f \approx 0.31$ for the road borders, the sensor reporting threshold of 0.06 will correspond to an actual physical distance of only 1.9 cm (0.06×0.31). As a consequence, the robot must be within 1.9 cm of a border before one sensor reports a value. In parallel, our kinematic reconstruction has a mean position error that in many cases can be comparable to, or larger than, the margin itself. This repeated positional error explains why sensor accuracy drops so sharply once the two components are combined.

Comparing our present approach with the SA-SBT methodology of Abdessalem et al. [22], it can be observed that in our case, the problem domain introduces significant sequential dependencies, while their framework predicts fitness values from a fixed set of five scenario parameters that have a direct relationship with the outcome. For example, a faster vehicle or a closer pedestrian straightforwardly reduces the minimum distance. In contrast, as demonstrated earlier, the ENPS-based controller operates over a long closed loop, where the same road geometry can lead to different outcomes depending on how the preceding trajectory aligns with a critical curve when the robot reaches it. This explains why road geometry alone predicts pass/fail outcomes well (AUC 0.93), but predicting the exact margin of failure by using continuous fitness functions does not transfer. Such a task would require the surrogate to account for the accumulated state of a long control loop, not just the final outcome. In parallel, Abdessalem et al. adopted neural networks over classical regression for their scenario space, citing their superior capacity for complex input spaces. Moreover, their outcome prediction remains driven by the static initial conditions of the scenario, rather than an evolving trajectory. This distinction outlines a criterion for choosing between different surrogate strategies in SA-SBT.

A critical consideration in SA-SBT is that the cost of generating training data must not outweigh the downstream savings. In the proposed approach, the net savings are preserved, for several reasons. First of all, the training set (1200 runs) does not represent an additional cost that is specific to the surrogate. More precisely, the verdicts of the runs from the training remain valid, so the computational savings accumulate across every subsequent test scenario. Moreover, once trained, the classifier evaluates a scenario in a negligible time

compared to a simulation, as scoring all 600 validation scenarios takes a fraction of a second, excluding the extraction of the road descriptors. Another argument is that the costs of the executions are asymmetric. As mentioned in Test Campaign, the analysis of the simulation time across all the runs indicated that successful scenarios have an average of 285 simulated seconds, whereas OOB failures last 60 s on average. Consequently, at the selected threshold of 0.30, filtering reduces the simulations by 51% and the simulated time by 71%, while retaining 97% of the true failures. Extrapolating the results measured on the validation set and assuming that new roads are generated, for a campaign of 1000 roads (12,000 scenarios), the validation set gives an average cost of 189 simulated seconds per scenario, resulting in roughly 620 simulated hours if all scenarios are simulated. As a result, considering that the first 100 roads are simulated in Webots as the training set, the total savings would be 400 h. Regarding the gray-box strategy, as it naturally embeds the symbolic rules of the ENPS controller and calibrates only the two kinematic parameters, there is no training campaign required.

Drawing upon the findings presented in this study, we now summarise the answers to the three RQs introduced in Section 1:

RQ1. Can a surrogate model reduce the cost of the simulation-based testing campaign for an ENPS-based controller, while preserving most of its fault-detection capability?

Yes, at the selected threshold of 0.30, the classifier avoids 307 of the 600 validation simulations (51%) and 71% of the simulated time, while retaining 219 of the 226 true failures (97%) on an independently generated validation set.

RQ2. To what extent can a surrogate model be constructed by exploiting the formal specification of an ENPS controller?

We showed that the ENPS traces containing velocity values allow accurate reconstruction of the robot trajectory through standard differential-drive kinematics. Although the controller kinematics and sensor behaviour can be accurately reconstructed from their symbolic definitions in isolation, composing them into a reactive closed loop introduces cumulative physical interactions. As a result, the surrogate cannot fully match the fidelity of the complete simulation to replace it entirely.

RQ3. How effectively does continuous fitness prediction transfer to long, reactive control loops?

The experimental phase revealed that in the current scenario, continuous fitness prediction does not transfer effectively to long, reactive control loops. We showed that the reconstructed position error is comparable to the sensor effective detection margin. This mismatch explains why predicting exact safety margins, which is effective in short, parametrically simple scenarios, proves unfeasible in our closed-loop configuration.

Even though the findings should be interpreted within the current setting, there are some factors that determine how they can be generalised. First of all, the controller uses ENPS model instances as core component. Consequently, the failure rate depends on the model capabilities. For example, Π_{M_1} fails on almost every test, whilst Π_{M_2} and $_{M_3}$ fail on a minority of them. Secondly, the classifier was trained and validated on road tests whose borders act like obstacles and the accuracy cannot be automatically associated with other types of scenarios (such as moving obstacles). Another factor is represented by the simulation platform and the simulated model, as sensor ranges and wheel slip were calibrated for the simulated E-puck in Webots and did not address the gap between simulation and physical robot. Moreover, the argument used for fitness prediction in reactive long control loops is plausible for similar reactive controllers, but is not tested in the current study.

8. Conclusions

In this paper, we investigated the application of SA-SBT to an obstacle avoidance robot controller based on ENPS. We built and evaluated a black-box classifier relying on road geometry and a gray-box surrogate reconstructed directly from the internal specification of the controller.

The black-box classifier returns a failure probability for a given road scenario. The probability is computed from geometric descriptors of the road together with the label of the ENPS model used by the robot controller, avoiding 51% of the Webots simulations (and 71% of the simulated time) at the selected threshold of 0.30, while retaining 97% of the true failures.

The gray-box surrogate instead returns a reconstructed trajectory, computed directly from the ENPS rules and a geometric model of the proximity sensors. It is capable of reproducing ground-truth motion to within a few millimetres and sensor readings with over 70% accuracy when each component is evaluated in isolation, but the composed closed-loop system does not reach the required positional accuracy, as previously explained. The gray-box surrogate was deliberately designed as a purely analytical formulation rather than a learned model, as our objective was to evaluate how far the controller explicit symbolic structure alone could be exploited. The controller used in this study along with simulation scenarios and their associated results are available at [42].

As future work, we aim to extend the evaluation to more diverse testing scenarios (e.g., obstacle arenas), further investigating the improvement of the closed-loop under different operational conditions. Furthermore, we plan to explore how the broader SA-SBT taxonomy established by Nejati et al. applies to other classes of P systems beyond ENPS.

Author Contributions: Conceptualization, F.I. and R.T.B.; methodology, F.I. and R.T.B.; software, R.T.B.; validation, F.I. and R.T.B.; formal analysis, R.T.B.; investigation, R.T.B.; resources, R.T.B. and F.I.; data curation, R.T.B.; writing—original draft preparation, R.T.B.; writing—review and editing, F.I.; visualization, F.I.; supervision, F.I.; project administration, R.T.B. and F.I. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The entire research data is published at [42].

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Rad, O.L.; Brisan, C. Control algorithms in robot-assisted rehabilitation: A systematic review. *Appl. Sci.* **2025**, *15*, 9184. [CrossRef]
2. Tzafestas, S.G. Mobile robot control and navigation: A global overview. *J. Intell. Robot. Syst.* **2018**, *91*, 35–58. [CrossRef]
3. Solanes, J.E.; Gracia, L. Mobile robots: Trajectory analysis, positioning and control. *Appl. Sci.* **2025**, *15*, 355. [CrossRef]
4. Salvato, E.; Fenu, G.; Medvet, E.; Pellegrino, F.A. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access* **2021**, *9*, 153171–153187. [CrossRef]
5. Stepien, L.; Thal, S.; Henze, R.; Nakamura, H.; Antona-Makoshi, J.; Uchida, N.; Raksincharoensak, P. Applying heuristics to generate test cases for automated driving safety evaluation. *Appl. Sci.* **2021**, *11*, 10166. [CrossRef]
6. Zhu, Y.; Wan Hasan, W.Z.; Harun Ramli, H.R.; Norsahperi, N.M.H.; Mohd Kassim, M.S.; Yao, Y. Deep reinforcement learning of mobile robot navigation in dynamic environment: A review. *Sensors* **2025**, *25*, 3394. [CrossRef] [PubMed]
7. Song, B.; Li, K.; Orellana-Martín, D.; Pérez-Jiménez, M.J.; Pérez-Hurtado, I. A survey of nature-inspired computing: Membrane computing. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–31. [CrossRef]
8. Păun, G.; Păun, R. Membrane computing and economics: Numerical P systems. *Fundam. Informaticae* **2006**, *73*, 213–227.

9. Bobe, R.T.; Gheorghe, M.; Ipate, F.; Niculescu, I.M. Numerical P Systems: Variants and Applications. In Proceedings of the 24th Conference on Information Technologies—Applications and Theory (ITAT 2024), Drienica, Slovakia, 20–24 September 2024; pp. 121–128.
10. Zhang, L.; Xu, F.; Xiao, D.; Dong, J.; Zhang, G.; Neri, F. Enzymatic numerical spiking neural membrane systems and their application in designing membrane controllers. *Int. J. Neural Syst.* **2022**, *32*, 2250055. [CrossRef] [PubMed]
11. Pavel, A.B.; Buiu, C. Using enzymatic numerical P systems for modeling mobile robot controllers. *Nat. Comput.* **2012**, *11*, 387–393. [CrossRef]
12. Bobe, R.T.; Gheorghe, M.; Ipate, F.; Niculescu, I.M. Test-driven simulation of robots controlled by enzymatic numerical P systems models. In *Proceedings of the International Conference on Simulation Tools and Techniques*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 56–69.
13. Zhang, Z.; Su, Y.; Pan, L. The computational power of enzymatic numerical P systems working in the sequential mode. *Theor. Comput. Sci.* **2018**, *724*, 3–12. [CrossRef]
14. 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* **2026**, *102*, 45–57. [CrossRef]
15. Michel, O. Cyberbotics ltd. webots™: Professional mobile robot simulation. *Int. J. Adv. Robot. Syst.* **2004**, *1*, 5. [CrossRef] [PubMed]
16. Farley, A.; Wang, J.; Marshall, J.A. How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion. *Simul. Model. Pract. Theory* **2022**, *120*, 102629. [CrossRef]
17. 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, Pittsburgh, PA, USA, 9 May 2022; pp. 43–46.
18. Humeniuk, D.; Khomh, F.; Antoniol, G. A search-based framework for automatic generation of testing environments for cyber–physical systems. *Inf. Softw. Technol.* **2022**, *149*, 106936. [CrossRef]
19. Harman, M.; Jia, Y.; Zhang, Y. Achievements, open problems and challenges for search based software testing. In *Proceedings of the 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*; IEEE: Piscataway, NJ, USA, 2015; pp. 1–12.
20. Harada, T.; Alba, E.; Luque, G. Energy and quality of surrogate-assisted search algorithms: A first analysis. In *Proceedings of the 2024 IEEE Congress on Evolutionary Computation (CEC)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–8.
21. Haq, F.U.; Shin, D.; Briand, L. Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 8–20 May 2022; pp. 811–822.
22. Ben Abdessalem, R.; Nejati, S.; Briand, L.C.; Stifter, T. Testing advanced driver assistance systems using multi-objective search and neural networks. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, 3–7 September 2016; pp. 63–74.
23. Nejati, S.; Sorokin, L.; Safin, D.; Formica, F.; Mahboob, M.M.; Menghi, C. Reflections on surrogate-assisted search-based testing: A taxonomy and two replication studies based on industrial adas and simulink models. *Inf. Softw. Technol.* **2023**, *163*, 107286. [CrossRef]
24. Pavel, A.; Arsene, O.; Buiu, C. Enzymatic numerical P systems-a new class of membrane computing systems. In *Proceedings of the 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*; IEEE: Piscataway, NJ, USA, 2010; pp. 1331–1336.
25. Zhang, Z.; Wu, T.; Păun, A.; Pan, L. Universal enzymatic numerical P systems with small number of enzymatic variables. *Sci. China Inf. Sci.* **2018**, *61*, 092103. [CrossRef]
26. Wang, X.; Zhang, G.; Gou, X.; Paul, P.; Neri, F.; Rong, H.; Yang, Q.; Zhang, H. Multi-behaviors coordination controller design with enzymatic numerical P systems for robots. *Integr. Comput.-Aided Eng.* **2021**, *28*, 119–140. [CrossRef]
27. McMinn, P. Search-based software testing: Past, present and future. In *Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*; IEEE: Piscataway, NJ, USA, 2011; pp. 153–163.
28. Aleti, A.; Moser, I.; Grunske, L. Analysing the fitness landscape of search-based software testing problems. *Autom. Softw. Eng.* **2017**, *24*, 603–621. [CrossRef]
29. Long, Y.; She, X.; Mukhopadhyay, S. Hybridnet: Integrating model-based and data-driven learning to predict evolution of dynamical systems. In *Proceedings of the Conference on Robot Learning*; PMLR: Cambridge, MA, USA, 2018; pp. 551–560.
30. Kezunovic, M.; Esmailian, A.; Manimaran, G.; Mehrizi-Sani, A. The use of system in the loop, hardware in the loop, and co-modeling of cyber-physical systems in developing and evaluating new smart grid solutions. In Proceedings of the Hawaii International Conference on System Sciences, Big Island, HI, USA, 4–7 January 2017.
31. Plummer, A.R. Model-in-the-loop testing. *Proc. Inst. Mech. Eng. Part I J. Syst. Control Eng.* **2006**, *220*, 183–199. [CrossRef]
32. Webots Reference Manual. Available online: https://cyberbotics.com/doc/reference/proto (accessed on 10 October 2026).

33. Webots Reference Manual for E-Puck. Available online: https://cyberbotics.com/doc/guide/epuck?version=R2022b (accessed on 10 October 2026).
34. 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*; Instituto Politécnico de Castelo Branco (IPCB): Castelo Branco, Portugal, 2009; pp. 59–65.
35. Couceiro, M.S.; Vargas, P.A.; Rocha, R.P. Bridging the reality gap between the Webots simulator and e-puck robots. *Robot. Auton. Syst.* **2014**, *62*, 1549–1567. [CrossRef]
36. Florea, A.G.; Buiu, C. Modelling multi-robot interactions using a generic controller based on numerical P systems and ROS. In Proceedings of the 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Târgoviște, Romania, 29 June–1 July 2017; pp. 1–6. [CrossRef]
37. Bobe, R.T.; Ipate, F.; Niculescu, I.M. Modelling and search-based testing of robot controllers using enzymatic numerical P systems. In *Proceedings of the 7th Symposium on Working Formal Methods, Bucharest, Romania, 21–22 September 2023*; Cheval, H., Leuştean, L., Sipoş, A., Eds.; OPA: Waterloo, NSW, Australia, 2023.
38. Florea, A.G.; Buiu, C. PeP—An Open-Source Software Simulator of Numerical P Systems and Numerical P Systems with Enzymes 2017. Available online: https://github.com/andrei91ro/pep (accessed on 10 August 2026).
39. Humeniuk, D.; Khomh, F. Ambiegen at the sbft 2024 tool competition-cps-uav track. In Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing, Lisbon, Portugal, 14–16 April 2024; pp. 69–70.
40. Gambi, A.; Jahangirova, G.; Riccio, V.; Zampetti, F. SBST tool competition 2022. In Proceedings of the 15th Workshop on Search-Based Software Testing, Pittsburgh, PA, USA, 9 May 2022; pp. 25–32.
41. Webots Reference Manual. Available online: https://cyberbotics.com/doc/reference/index (accessed on 15 September 2026).
42. Github Repository. Available online: https://github.com/radubobe/Mobile-Robots-Optimisation/tree/master/SA-SBT%20ENPS (accessed on 25 October 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.