Journal of Membrane Computing
https://doi.org/10.1007/s41965-026-00230-6

RESEARCH PAPER

Sensor-driven evolutionary optimisation of mobile robot controllers using enzymatic numerical P systems

Radu Traian Bobe[1] · **Marian Gheorghe**[1,2] · **Florentin Ipate**[1]

Received: 2 February 2026 / Accepted: 20 August 2026
© The Author(s) 2026

Abstract
In recent years, the application of membrane computing within the field of robotics has been widely investigated, providing a bio-inspired alternative to traditional control paradigms. This paper proposes an approach that combines Enzymatic Numerical P Systems (ENPS) with Non-dominated Sorting Genetic Algorithm (NSGA-II), a well-known multi-objective evolutionary optimisation algorithm, as core components of a real-time obstacle avoidance robot controller. The ENPS is employed as the computational framework due to its inherent parallelism, modular structure and suitability for sensor-driven control, enabling a natural mapping between sensory inputs and motor actions. To enhance real-time adaptability, we design NSGA-II to interact with the robot controller during the simulation, optimising the key parameters of the equations that define the robot behaviour, which are implemented as production functions within the ENPS model instance. This improvement leads not only to passing the testing scenarios used in designing the robot controller, but also to the robot speed and trajectory optimisation. In addition, the paper provides a comparative analysis, employing non-parametric testing to benchmark the proposed NSGA-II-based controller against static-weight alternatives. A comprehensive suite of quantitative performance metrics, including execution time, traveled distance or memory consumption is utilised to validate the superiority of the proposed approach.

Keywords Membrane computing · Genetic algorithms · Robot controllers · Multi-objective algorithms · Enzymatic numerical P systems · Simulation

1 Introduction

Robot vehicles have been the subject of many large-scale experiments from the second half of the last century to the present day. As a natural consequence, the nature and importance of the tasks that these *cyber-physical systems* [1] have to fulfill has changed significantly. One of the most common use cases in robotics remains the obstacle avoidance scenario, where the mobile robot moves inside an unknown environment with the objective of reaching a target. Taking into account that the large majority of industrial areas use robots that must be capable of performing obstacle avoidance [2] and movement in dynamic environments, this subject still retains a high degree of relevance and innovative approaches that can enhance *robot controller* performance should be considered. To achieve this type of mission, the sensor system of the robot must be employed efficiently. The availability of a performant robot operating in unknown environments not only reduces the amount of work that a human would perform, but also provides a solution to circumstances that may be faced with human decision errors. The obstacle avoidance problem requires real-time synthesis of sensor data into precise motor commands. In this context, *Numerical P systems* [3], a well-known class of membrane systems, emerge as a compelling computational framework, motivated by several advantages. First of all, the inherent parallelism allows for the simultaneous execution of production functions across multiple computational units. For obstacle avoidance, this means that data from proximity sensors can be processed in parallel, significantly reducing the latency between per-

✉ Radu Traian Bobe
radu.bobe@s.unibuc.ro

Marian Gheorghe
m.gheorghe@bradford.ac.uk

Florentin Ipate
florentin.ipate@unibuc.ro

[1] Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Str Academiei 14, 010014 Bucharest, Romania

[2] Faculty of Engineering and Informatics, University of Bradford, Bradford, West Yorkshire BD7 1DP, UK

Published online: 29 September 2026

ception and action. Moreover, the mathematical flexibility enhanced by the use of polynomial functions, as well as the computational power of Numerical P systems (NP systems) have been demonstrated. [4]. Our latest article [5] presented a methodology developed in a test-driven manner that helped us design robot controllers using one of the most well-known variants of NP systems [6], called *Enzymatic Numerical P systems* (ENPS). ENPS [7, 8] introduce a sophisticated layer of control where the execution of a production function is conditional upon the value of the associated enzyme variable, supporting the parallel execution of multiple rules within the same membrane. ENPS can be used as a rigorous framework in modeling robot controllers, offering high control of the execution enhanced by the integration of the enzymes [9, 10]. The effective integration of this type of numerical P system within models designed for obstacle avoidance can be achieved via PeP, a simulator for running NP systems [11].

The operational limits of an obstacle avoidance controller should be a priority issue when discussing the reliability of these systems in different environments. As the performance of controllers varies depending on the environment in which they are used, one approach to consider is diversifying the testing scenarios. Search-based software testing (SBST) [12, 13] can be a solution to this problem by providing a vast and diversified testing environment tailored to the requirements, which can be specified using configurable parameters within the search-based algorithm. In search-based testing, meta-heuristic techniques are used to find relevant solutions for predefined objectives [14]. The quality of the generated solutions is guided by fitness functions that can estimate how efficient the resulting test case is. Search-based testing has been applied in test generation for different types of cyber-physical systems [15–17]. The definition of simulation scenarios for systems, such as autonomous cars, constitutes an ongoing area of research in the field and compelling examples of search-based software testing tools were presented at SBST competition [18], where AmbieGen [19] was remarked upon.

In our previous work, several enzymatic numerical P systems capable of exchanging information with the Robot Operating System (ROS) were developed. Our models were based on the initial example presented in [20] which was conducted on an educational robot called *E-puck* [21], which is represented in a virtual environment provided by Webots [22], a robot simulation tool.

Within the initial approach, the robot controller receives the proximity sensor values from E-Puck and transmits them to these enzymatic numerical P system models that associate the proximity sensor values with some empirically chosen constant parameters designed to achieve the desired behaviour, called *weights*. The empirical manner of selecting these weight values is labor-intensive, as it requires numerous trials and failures to find the right combination of values. Moreover, to obtain more natural robot behaviour, such as maintaining a centered trajectory, the model structure has become complex and difficult to adjust. In this way, to exclude the empirical component, we elaborated a refined heuristic and proposed a parametrisation method based on *linear regression* [23]. In brief, this consisted of finding the relationship between the speed of the wheel, which in terms of linear regression can be considered the dependent value, and the proximity sensor values (independent values). In addition to a more elaborate heuristic and good behaviour during the test phase, we also obtained a simplified model structure. However, even if the weights were chosen via linear regression and the empirical component was excluded, the effective values were given to the model as constants, so their values remained unchanged regardless of the circumstances.

On the basis of the future research directions mentioned in our previous paper, in this article we present a way to improve the behaviour of the ENPS-based robot controllers using real-time multi-objective optimisation. Consequently, we propose an adaptive solution that exhibits adaptability across varying scenarios. This is accomplished by integrating in the controller the *Non-dominated Sorting Genetic Algorithm II (NSGA-II)*, a proven multi-objective evolutionary optimisation algorithm [24]. This upgraded variant of the robot controller replaces fixed weight assignment with adaptive parameter selection, which is relevant in situations where the number of obstacles in a scenario is changing. Moreover, the controller eliminates the need for manual weight assignment across test categories by implementing an integrated multi-objective optimisation algorithm that autonomously fine-tunes parameter values that are used as variables in the ENPS instances. Notably, this optimisation framework is designed for seamless integration with the ENPS models using empirical weights that were developed in our previous research.

The novelty of this work lies in three major aspects:

- we propose a hybrid control architecture in which ENPS is used as the core computational framework for behaviour generation, while NSGA-II dynamically adapts the sensor–motor weights
- the proposed approach demonstrates the practical role of several ENPS model instances developed in a "test-driven" manner as the core component of a reactive obstacle avoidance controller
- the work introduces online evolutionary adaptation within an ENPS-based control architecture, enabling dynamic adjustment of control parameters during navigation. Unlike classical approaches where evolutionary algorithms are used offline to tune controller parameters, the proposed method embeds NSGA-II directly into the

perception–decision loop, allowing continuous adaptation to changing environmental conditions

To demonstrate the quantitative performance and the functional integration of the proposed solution, several evaluation metrics are used. Specifically, we investigate the controller's efficiency through an analysis of execution time, memory consumption, and traveled distance, alongside stability indicators, such as average speed variation and obstacle clearance. These metrics allow us to evaluate how the evolutionary-defined parameters enhance the ENPS framework's responsiveness.

The study is organised as follows. Section 2 highlights the context of our previous work in this domain and some existing links from the literature between the central concepts of this study. Section 3 introduces the definitions of enzymatic numerical P systems as well for genetic algorithms and NSGA-II. Section 4 explains the functioning of the robot controller, including technical details regarding the implementation of the multi-objective optimisation algorithm. Section 5 illustrates various experimental approaches along with relevant results, demonstrating the impact of the NSGA-II optimisation. Section 6 discusses the role of baseline comparisons in obstacle avoidance evaluation, presenting a brief comparative analysis between the proposed controller and a classical Artificial Potential Fields approach. Finally, Sect. 7 states the conclusions and some possible future directions of research.

2 Related work

The advancement of membrane computing has led to the development of sophisticated frameworks, such as variants of numerical spiking neural P systems (NSN P systems), which have demonstrated significant potential in real world applications [25]. Recent research has further explored the capabilities of these systems for continuous optimisation problems [26]. In addition, variants of numerical P systems have recently been utilised to solve complex classification problems [27].

The integration of membrane computing within robotic architectures represents a well-established area of research. While early investigation focused on the theoretical feasibility of using P systems for path planning [28], more recent literature has addressed the inherent complexities of real-world applications [29], such as hardware implementation [30] or modeling controllers for walking biped robots [31]. This paper aims to extend the application of ENPS to specific obstacle avoidance scenarios where enzymatic regulation provides a critical advantage in control stability.

As stated in Sect. 1, our previous article [5] proposed a methodology for building simulators for robot controllers that are based on different enzymatic numerical P system models. Starting from the limitation exposed in the test stage by each model, we worked in a *test-driven* manner [32], examining model performance under different circumstances and then adding improvements. All these incremental improvements led us to five different models involved in our final experiments. The experiments have shown that both the empirical weights and linear regression-based models obtained good results in terms of passing different types of test scenarios. In other words, finding a good combination of constant weights during the test can provide satisfying results, as the test is passed. Starting from this assumption, we focused on an improved solution from other perspectives, such as those mentioned in Sect. 1 and detailed later in this article and not just passing or failing the test suite. In this way, we introduced a dynamic manner of finding and assigning the weights during a test, by computing them using a genetic multi-objective optimisation algorithm which is called inside the robot controller during the simulation.

The association between genetic algorithms and robotics is a compelling topic with multiple approaches in the literature. Manikas et al. [33] provided an overview of applying genetic algorithms to autonomous robot navigation. A robot similar to E-Puck, called Khepera, was the subject of some experiments carried out by Nordin and Banzhaf [34] on the use of genetic algorithms in the evolution of real-time obstacle avoidance behaviour. The path planning NP-hard problem has been a classical subject in robotics for many years. In 1993, Ahuactzin et al. [35] presented a study regarding robot motion planning using genetic algorithms. In 2018, 25 years later, the problem of obstacle avoidance in unknown environments was addressed by Kwaśniewski and Gosiewski in [36]. They presented both the advantages and disadvantages of using GA for path finding. Additionally, in 2018, Mao et al. constructed a robot space obstacle avoidance model based on genetic evolutionary adaptive guidance control [37].

Another interesting approach was presented by Binch et al. in [38], who developed a context dependent framework with an incorporated genetic algorithm that makes parameter tuning by means of continuous optimisation. From some point of view, their developed framework achieves a "black-box" optimisation, as the robot operating system navigation methods are independent of the framework.

As the need for an optimal parameter tuning in various environments is obvious, different methods have been studied for application in robotics. Genetic algorithm tuning methods are promising, as they can provide better results than other traditional methods for controller tuning, as demonstrated in [39–41].

Our proposed robot controller represents a hybrid adaptive strategy that combines evolutionary multi-objective optimisation with membrane computing-based dynamics for real-time weight adaptation. The proposed approach does

not align with other initiatives specifically designed for path planning; rather, it focuses on the dynamic adaptation of robot behaviour regardless of the environmental characteristics or the number of encountered obstacles. To achieve this, the genetic algorithm, which provides a powerful method for finding near-optimal solutions adapted to specific behavioural needs, is integrated into the robot controller's logic, searching for adequate values at each step of the simulation.

Reinforcement Learning (RL) is another paradigm that has shown promising results in the field of robotics [42, 43]. In such algorithms, performance heavily depends on the quality of reward shaping and availability of training episodes [44]. Reinforcement Learning is ideal when the environment can be simulated extensively ahead of deployment and is less effective when the environment changes frequently. In comparison, an implementation based on evolutionary algorithms is well-suited for real-time weight adaptation, being capable of balancing multi-objectives, such as obstacle avoidance and movement smoothness. Moreover, it requires no training phase, as adaptation begins immediately. Also, evolutionary methods are generally more robust to such noise because selection operates on a diverse population rather than a single accumulating reward signal.

In terms of scalability, Reinforcement Learning excels in tasks involving large perceptual inputs (e.g. visual navigation) due to the representational power of neural networks [45]. However, this scalability comes at the cost of memory consumption, computational load and the need for specialized hardware, such as GPUs. In contrast, an evolutionary based approach requires a modest population, making it more suitable for embedded robotic systems with limited processing resources. Although evolutionary optimisation does not scale as efficiently to very high-dimensional parameter spaces, it remains highly effective in scenarios where the controller consists of a structured, low-dimensional set of interpretable parameters, as in the current scenario discussed in this approach.

Overall, the controller introduced in this paper offers complementary strengths to other widely adopted strategies in robotics. It provides rapid and efficient adaptation, being suitable for dynamic obstacle avoidance, whereas Reinforcement Learning remains more appropriate for tasks requiring complex, long-horizon decision making. The proposed method, therefore, represents a pragmatic alternative for real-time robotic applications in which immediate sensory-driven adaptation is essential.

3 Preliminaries

3.1 Enzymatic numerical P system definition

Being already mentioned that the robot controller behaviour is guided by instances of the enzymatic numerical P system (ENPS) model, we first introduce the ENPS definition. The (enzymatic) numerical P systems [46] are classes of membrane systems that only inherit the membrane structure from them, employing *numerical variables* in the compartments of a cell-like architecture. These variables are initialised with input values provided at the beginning of the computation. They evolve over time according to programs defined as *evolution rules*. At each computation step, the enabled rules associated with different membranes are applied concurrently, allowing variables to be updated in a parallel and distributed manner across the system. The resulting system with membranes delimited as compartments is controlled by a global clock through discrete time units. Unlike classical numerical P systems, enzymatic numerical P systems introduce enzymes as control mechanisms within evolution programs. These enzymes regulate the application of production functions, as a program is executed only when the value of the associated enzyme is strictly greater than the minimum value of the variables involved in the production function. This enzymatic conditioning enables finer control over numerical transformations, making ENPS particularly suitable for robot control algorithms [47].

The enzymatic numerical P system is defined by the tuple:

$$\Pi = (m, H, \mu, (\mathrm{Var}_1, \mathrm{Pr}_1, \mathrm{Var}_1(0)), \ldots, (\mathrm{Var}_m, \mathrm{Pr}_m, \mathrm{Var}_m(0))) \quad (1)$$

where:

- $m \geq 1$ is the degree of the system Π (the number of membranes);
- H is an alphabet of labels;
- μ is the membrane structure (a tree);
- Var_i is a set of variables from membrane i, $1 \leq i \leq m$;
- $\mathrm{Var}_i(0)$ represent 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 $\mathrm{Pr}_{l_i,i}$, $1 \leq l_i \leq m_i$ has one of the following forms:

  (i) non-enzymatic

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

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 $v_1, v_2 \ldots v_{m_i}$ belong to the compartment containing the programs, and to its upper and inner compartments, for a specific compartment 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

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

Remark 1 The system evolves over time by executing all the programs that satisfy condition 4 (the enzymatic control). This occurs in parallel in all compartments.

When the program 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 $v_1,\ldots,v_n$, proportional to coefficients $c_{1,i},\ldots,c_{m_i,i}$, where $c_{j,i} \in \mathbf{N}^+$ and the received values will be $q_{l_i,i}(t)\cdot c_{1,i},\ldots,q_{l_i,i}(t)\cdot c_{m_i,i}$.

Remark 2 The value of each of the variables from time $t-1$, occurring in the production functions is *consumed* (reset to zero) and its new value is the sum of the proportions distributed to variable through the repartition protocols, if it appears in them, or remains at value zero if not.

3.2 Genetic algorithms

Genetic algorithms (GA) are metaheuristic search algorithms introduced by Holland [48] as a component of the set of biologically inspired search methods, called *evolutionary algorithms* (EA) [49]. A GA finds near-optimal solutions by repeatedly modifying a population of individual solutions (named "*chromosomes*") using a set of genetically inspired operators: *selection, crossover* and *mutation* [50].

The *selection* operator, also known as reproduction, selects the fittest individuals and enables them to transmit their genes to future generations. Techniques as *roulette wheel selection*, *tournament selection* and *rank-based selection* are employed to determine the individuals selected for the next generation [51]. The *crossover* or recombination operator is used to exchange components (named "*genes*") of individuals, creating a new one (named "*offspring*"). Methods of crossover include *single-point crossover*, *N-point crossover* or *uniform crossover* [52]. The *mutation* operator is used to insert random genes into the offspring. The purpose of this is to maintain the diversity in population and also to prevent the loss of information that occurs as a consequence of premature convergence. The entire process is guided by an evaluation function, known as *fitness function*, which returns a score that indicates how well an individual performs. In this way, after the population is evaluated, individuals with better fitness scores have more chances to reproduce in the next generations. The algorithm is considered to converge to a set of solutions when there is no significant difference between the current offspring and that of previous populations. As a preliminary conclusion, we can consider that genetic algorithms explore the search space for possible solutions, while exploiting selection techniques.

3.3 NSGA-II: Non-dominated Sorting Genetic Algorithm

Non-dominated Sorting Genetic Algorithm (NSGA-II) is a commonly applied multi-objective optimisation algorithm developed by Deb et al. in 2000 [24]. This algorithm was proposed as an improvement of the first variant [53], addressing some limitations, particularly with respect to computational efficiency and ensuring diversity of the optimal solutions. This multi-objective approach focuses on finding a set of solutions that represents the best trade-off among multiple objectives. The entire group of the possible solutions is organised as *Pareto fronts*, used to classify a population. The Pareto fronts are obtained after comparing individuals based on their objective values. After the population is sorted, within each front, the *crowding distance* is computed to promote diversity among the individuals [54]. A higher crowding distance is equivalent to better diversity and is calculated as the sum of the normalized distances between adjacent individuals from the same front.

The next step of the algorithm involves the selection of individuals based on their front rank and crowding distance, followed by the classical operations, such as crossover and mutation, to create new offspring. The process will repeat until a termination criterion is met, the most common being when a fixed number of generations is reached or a convergence of solutions is encountered, meaning that the population's fitness improvement stagnates. In the end, the final population is considered to be the *Pareto optimal solution*. To encapsulate the explanation given, we provide a step by step summarization of the NSGA-II algorithm, as follows:

Returning to our research area, after an analysis of the current literature, we observed that NSGA-II is occasionally employed in problems involving different types of cyber-

Algorithm 1 NSGA-II working mechanism

```
Random initial population
Evaluate fitness for each individual
Non-dominated sorting
Compute crowding distance
repeat
    Tournament selection
    Crossover and mutation
    Evaluate fitness for each individual
    Non-dominated sorting
     Compute crowding distance
     Select individuals
until the termination criterion is met
return Pareto optimal front
```

physical systems [55–57]. In our current approach, we find NSGA-II as a powerful solution for dynamic weight generation, as the algorithm can simultaneously optimise multiple objectives that can be conflicting under some circumstances. Also, NSGA-II can provide adaptability to different robot configurations, as the algorithm parameters can be easily adjusted. In Sect. 4, a more in-depth analysis of choosing NSGA-II is presented, along with the algorithm overview in our context.

4 Design of the algorithm and controller: structure and functionality

It has been demonstrated in our previous work that constant weights (empirically chosen or after a more complex heuristic, such as linear regression) can lead the robot controller to achieve good results in terms of passing the test scenarios. However, the ENPS model had a proven central role, as the initial model was not able to pass the test suite. In addition, the single metric that was considered was passing or failing the test. In this section, we detail our NSGA-II implementation as well as the way it interacts with the robot controller based on different ENPS models. Our goal is to optimise the entire functionality of the robot controller, taking into account some other characteristics that should be achieved, additional to just passing the tests. Since an optimisation should account multiple objectives concurrently, the use of a multi-objective optimisation algorithm seems to be a natural decision.

4.1 Enzymatic numerical P system models

As stated, we developed multiple instances of enzymatic numerical P system models that are used inside the robot controller and dictate the behaviour. In this study, we highlight some of them that were considered the most relevant to be included in the testing phase.

It is important to note that the proximity sensors can detect obstacles within approximately 4 cm range. A smaller positive reading corresponds to a closer distance, with the mention that for distances greater than the robot's range, the sensors will return zero. The values are then passed to the model that calculates the linear and angular velocity.

Before introducing the main improvements of each instance, we reiterate that all subsequent models were developed in our previous research (originally employing static weights), following a "test-driven" approach. Consequently, each model was iteratively designed to address specific performance or behavioural limitations observed during the evaluation of its predecessor. The present article contributes by proposing an evolutionary algorithm-based approach designed to dynamically optimise the weight parameters within the ENPS framework. Therefore, an ENPS model which would otherwise rely on empirically predetermined fixed weight sets can be seamlessly integrated as the core computational component of the controller, while benefiting from adaptive parameter optimisation. This approach thereby eliminates the need for extensive pre-simulation investigations previously required to determine suitable weight values.

4.1.1 Basic model with rotation

The first model we used in our experiments with the NSGA-II based robot controller is called *Basic Model with Rotation*, as it was created after noticing the limitations of the initial model tested in [58], referred to as Π_{M_1}. The main issue with Π_{M_1}, which caused it to fail the majority of road tests, was the insufficient rotation in the opposite direction when facing an obstacle. This can be explained by the fact that the speed of the wheel on the obstacle side increased only slowly, due to the selected weight values. The *Basic Model with Rotation*, Π_{M_2}, has a more complex membrane structure, with an additional compartment which is used to compute the product of the travel speed and the sum of the weights. By incorporating this newly defined compartment, the robot behaviour improved. Next, we introduce the equations describing how *weightLeft* and *weightRight* are obtained and their contribution to *leftSpeed* and *rightSpeed*:

$$weightLeft = \sum_{i=1}^{n} weightLeft_i \cdot \text{prox}_i \tag{6}$$

$$weightRight = \sum_{i=1}^{n} weightRight_i \cdot \text{prox}_i \tag{7}$$

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

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

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

Then, the motor speeds, denoted by *leftSpeed* and *rightSpeed* can be defined as follows:

$$\begin{aligned} leftSpeed &= cruiseSpeed \cdot weightLeft \\ &\quad + f(weightLeft) \cdot cruiseSpeed \end{aligned} \tag{9}$$

$$\begin{aligned} rightSpeed &= cruiseSpeed \cdot weightRight \\ &\quad + f(weightRight) \cdot cruiseSpeed \end{aligned} \tag{10}$$

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

The model instance is defined as follows:

$$\begin{aligned} \Pi_{M_2} = (&m, H, \mu, (\mathrm{Var}_1, \mathrm{Pr}_1, \mathrm{Var}_1(0)), \\ &\ldots, (\mathrm{Var}_m, \mathrm{Pr}_m, \mathrm{Var}_m(0))) \end{aligned} \tag{11}$$

where:

- $m = 3k + 3, k = 6$;
- $H = \{s, w, s_c\} \cup \bigcup_{i=1}^{k} \{c_i, s_i, w_i\}$;
- $\mu = [[[[]_{s_1}[]_{w_1}]_{c_1} \ldots [[]_{s_k}[]_{w_k}]_{c_k}[]_{s_c}]_w]_s$;
- $\mathrm{Var}_s = \{x_{s_l}, x_{s_r}\}$, $\mathrm{Var}_w = \{x_{w_l}, x_{w_r}, e_w\}$, $\mathrm{Var}_{s_c} = \{x_{s_c}\}$,
  $\mathrm{Var}_{c_i} = \{x_{c_i,s_l}, x_{c_i,s_r}, x_{c_i,w_l}, x_{c_i,w_r}, e_{c_i}\}, 1 \le i \le k$,
  $\mathrm{Var}_{s_i} = \{x_{s_i,i}\}, 1 \le i \le k$,
  $\mathrm{Var}_{w_i} = \{x_{w_i,w_l}, x_{w_i,w_r}, e_{w_i}\}, 1 \le i \le k$;
- $\mathrm{Var}_i(0) = 0, 1 \le i \le m$;
- $\mathrm{Pr}_s = \{0 \cdot x_{s_l} \cdot x_{s_r} \to 1|x_{s_l} + 1|x_{s_r}\}$;
  $\mathrm{Pr}_w = \{x_{s_c} \cdot x_{w_l} + f(x_{w_l}) \cdot x_{s_c}|_{e_w} \to 1|x_{s_l},$
  $x_{s_c} \cdot x_{w_r} + f(x_{w_r}) \cdot x_{s_c}|_{e_w} \to 1|x_{s_r}\}$;
  $\mathrm{Pr}_{s_c} = \{x_{s_c} \to 1|x_{s_c}\}$;
  $\mathrm{Pr}_{c_i} = \{x_{c_i,s_l} \cdot x_{c_i,w_l}|_{e_{c_i}} \to 1|x_{w_l},$
  $x_{c_i,s_r} \cdot x_{c_i,w_r}|_{e_{c_i}} \to 1|x_{w_r}\}, 1 \le i \le k$;
  $\mathrm{Pr}_{s_i} = \{3x_{s_i,i} \to 1|x_{s_i,i} + 1|x_{c_i,s_l} + 1|x_{c_i,s_r}\},$
  $1 \le i \le k$;
  $\mathrm{Pr}_{w_i} = \{2x_{w_i,w_l}|_{e_{w_i}} \to 1|x_{w_i,w_l} + 1|x_{c_i,w_l},$
  $2x_{w_i,w_r}|_{e_{w_i}} \to 1|x_{w_i,w_r} + 1|x_{c_i,w_r}\}, 1 \le i \le k.$

The meaning of the variables from the model is outlined below:

- x_{s_l} and x_{s_r} from compartment s represent *leftSpeed* and *rightSpeed*, the sum of the products are accumulated in s after applying the rules from the membranes Pr_{c_i}, where $1 \le i \le k$;
- x_{w_l} and x_{w_r} from compartment w represent the *weightLeft* and *weightRight*, computed as described in Eqs. 6 and 7;
- x_{s_c} from the compartment s_c is *cruiseSpeed*;
- variables x_{c_i,s_l} and x_{c_i,s_r} from compartment c_i represent the sensor values multiplied inside compartments s_i, whilst x_{c_i,w_l}, x_{c_i,w_r} are the values of $weightLeft_i$ and $weightRight_i$ derived from compartments w_i;
- each pair of weights, $weightLeft_i$, $weightRight_i$ is denoted by x_{w_i,w_l}, x_{w_i,w_r} and resides in compartments w_i, $1 \le i \le k$;
- for each proximity sensor, prox_i, a compartment is defined, namely s_i, containing a single variable, $x_{s_i,i}$, $1 \le i \le k$.

One fact that may be confusing is the addition of x_{s_l} and x_{s_r} multiplied by zero in the production function from the compartment s. We adopted this approach that is equivalent to resetting the speeds, to consume the previous values of *leftSpeed* and *rightSpeed* at each simulation step.

The *weight* compartment w has two programs, responsible of aggregating weight contributions and applying the *cruiseSpeed*. It multiplies accumulated *weightLeft* and *weightRight* by *cruiseSpeed*, with a fallback mechanism (adds *cruiseSpeed* if weight is 0), as outlined in Eqs. 9 and 10, then sends results to parent speed compartment s, using the enzyme.

One can notice that the *cruiseSpeed* compartment s_c contains a self-replicating program that maintains the *cruiseSpeed* value and makes it available to the parent compartment w. By analysing Eqs. 6 and 7, it can be observed that each sensor has a weighted contribution to *weightLeft* and *weightRight*. This contribution is achieved from the compute compartments $c_i, 1 \le i \le n$, where n represents the number of sensors. Moreover, compartments 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. A similar task is fulfilled by compartments 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 corresponding compute compartment c_i.

4.1.2 Refined model

The previous model has led to good performance in passing the tests. However, the "zig-zag" motion going from the proximity of a road border to the proximity of the other one conducted us to develop the *Refined Model*. As defined in [59], Π_{M_3} provides a more natural behaviour, recentering after avoiding the obstacles.

To achieve this enhanced behaviour, an additional membrane, designated as "*Direction*", has been integrated. The core benefit of this addition is that inside this membrane, a newly introduced "*state*" variable replicates a finite state machine within the production functions, consisting of the following states:

(a) state 0—the robot is moving in a straight line
(b) state 1—the robot is moving in the presence of an obstacle

(c) state 2—the robot is moving to approximately the center of the lane
(d) state 3—the robot is recentering on the lane.

The "*state*" variable acts as a coordinator between the different behavioural phases of the robot, conditioning the production rules that compute the left and right wheel speeds based on the weighted sensor inputs propagated from the surrounding membranes. This logic was integrated into the membrane structure while maintaining the same operational principles regarding the input and output variables. Specifically, as in Π_{M_2}, the inputs are proximity sensor values and weights and the outputs are the wheel speeds. Although the number of compartments and variables increased, the robot's behaviour exhibited a visible improvement, becoming significantly more natural in obstacle avoidance scenarios.

4.1.3 Linear regression-based model

Our latest model, *Linear Regression-Based Model* was motivated by a different perspective. If the previous models led to advancements in terms of robot behaviour, this model was developed to substitute the empirical manner of choosing the weights with a more elaborate heuristic.

To develop this model, referred to as Π_{M_5} (Π_{M_4} is an extension of the *Refined Model*, adjusted for some particular cases described in our previous article, but of no interest for the current approach), we defined a linear relationship between the actual speed of the wheels and the values of the proximity sensors.

Using linear regression, Π_{M_5} became the model with the simplest structure. Let us present the relationship underlying the linear regression:

$$leftSpeed = a_0 + \sum_{i=1}^{n} a_i \cdot \text{prox}_i \tag{12}$$

$$rightSpeed = b_0 + \sum_{i=1}^{n} b_i \cdot \text{prox}_i. \tag{13}$$

The above two first-degree polynomials highlight the relationship between the instantaneous wheel speeds (dependent variables) and the proximity sensor readings, prox_i (independent variables).

The performance of Π_{M_5} is directly influenced by the values of a_i and b_i parameters, collected after running several simulations in Webots. These values are persisted on each simulation step using the enzymes. The parameters are then used to simultaneously execute the formulas presented in Eqs. 12 and 13 inside the compartments.

Remark 3 The formal definition of each model instance, as well as the comparison between their structure and performance are presented in detail in our previous article [5].

4.2 NSGA-II configuration

Prior to detailing the integration of the genetic algorithm into the logic of the robot controller, we consider it necessary to describe some of the implementation features.

First of all, an individual represents a set of weights that will be passed to the model. Each proximity sensor has a pair of weights (*weightLeft, weightRight*) with balanced values in order to ensure an adequate behaviour. In addition, the importance of balancing the values inside each pair is argued by the way that *leftSpeed* and *rightSpeed* are computed, as presented in Sect. 4.1. For example, if an obstacle is detected by a proximity sensor from the right side, the right wheel needs to slow down, whilst the left wheel speeds up. This results in a turning motion in the opposite part of the obstacle. Opposing weight distributions can facilitate *differential steering*, contributing to the stability and natural motion of the robot. As concluded from above, the number of *genes* per individual will be determined by the number of the proximity sensors, enhancing the flexibility provided by our approach. Our examples used 6 proximity sensors placed radially on our robot, E-puck. Furthermore, we used a *population size* of 50, meaning that each generation will contain 50 individuals. To balance the exploration and exploitation, we used a *mutation strength* of 0.5, along with a *number of genes to mutate* of 3. This implies that each mutation will change 3 randomly selected genes and will apply a perturbation from the interval $[-0.5, 0.5]$. We obtained these values after comparing the convergence speed and diversity of the individuals using different parameter values. To conclude, a larger *mutation strength* showed more diversity, but a slower convergence, while a small value resulted in faster convergence with a risk of limited diversity, which can lead to premature solution stagnation.

During the development phase, the collected data revealed a clear positive correlation between population size and computational resource consumption. Our analysis demonstrated that CPU utilization exhibits a consistent upward trend as population size increases. (Fig. 1) Specifically, the baseline population of 50 individuals yields a mean CPU utilization of 41.16%, while doubling the population to 100 individuals results in a 47.6% increase to 60.74% mean CPU utilization. This escalating pattern continues with a population of 150, where mean CPU utilization reaches 76.54%, representing an 86.0% increase relative to the baseline. The most substantial resource demands are observed at a population size of 500, where mean CPU utilization peaks at 95.67%, with an observable loss of precision and efficiency in results.

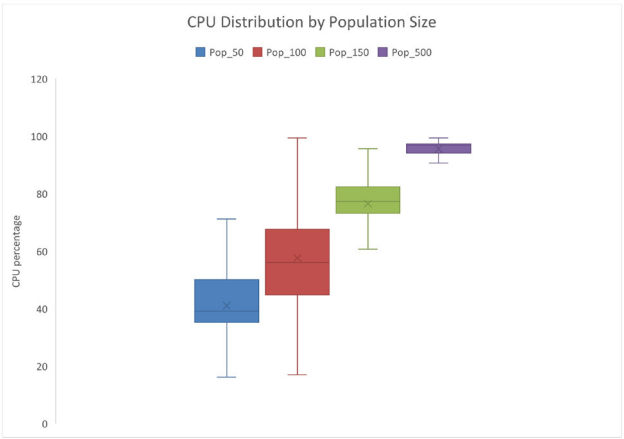


Fig. 1 CPU benchmarks across different population sizes



Fig. 2 Memory benchmarks across different population sizes

Similarly, memory consumption demonstrates a parallel upward trend, with median values increasing from approximately 2.5 MB for a population of 50 to approximately 5 MB for a population of 100, approximately 7.5 MB for a population of 150 and reaching approximately 22 MB for a population of 500, thereby exhibiting a comparable positive correlation between population size and memory utilization.

These findings underscore the critical importance of computational resource planning when scaling population-based algorithms or simulations.

Next, we outline all the parameters used during our experimental stage. The values were selected based on preliminary experiments. A systematic analysis demonstrating their robustness is provided in Sect. 5.2.

- Population size: 50
- Weights per individual: 12
- Mutation strength: 0.5
- Number of genes to mutate: 3
- Crossover type: uniform (50% of genes from each parent)
- Tournament size: 3.

4.3 Fitness functions

As stated before, the main goal of developing an NSGA-II algorithm to interact with the robot controller via ENPS models is to determine adequate sets of weights in a dynamic manner under different environmental circumstances during a test scenario. However, this brings up a question of significance to the entire discussion: what determines whether a weights set is good or not?

In addition to the previous discussions on the assignment of balanced values within pairs of weights associated with a sensor (that can be easily achieved using the empirical manner of finding weights), there are some other aspects to consider. First of all, proper values of weights should facilitate *avoiding collisions* in different environment setups. Based on the weights values, a collision can be avoided at different times, the movement of the robot in close proximity of the obstacle varying according to the weight values.

Another objective of the robot that should be considered is the maintenance of natural movements throughout the obstacle tests, hereafter referred to as *smoothness*. Therefore, the controller should prevent sudden changes caused by irregular moves. This undesired behaviour may be caused by abrupt changes of weights.

Lastly, weight *diversity* represents a fundamental consideration when discussing dynamic environments. The presence of the above-mentioned objectives suggests the need for diverse weight values depending on the test features, as achieving all these key characteristics with static weights is laborious. The *diversity* component can be achieved using genetic algorithms, as the returned solutions are evolved during several generations and a premature convergence can be avoided with proper configuration parameters.

Following an analysis of these objectives, we define two fitness functions which serve as the evaluation mechanism for each individual. We introduce the mathematical representation for each of them.

As stated before, the first objective is to encourage the robot to turn away from obstacles. Looking at the E-puck's sensors placement, we can affirm that *weightLeft* values for the left-sided sensors should be positive numbers large enough to influence the left wheel speed. At the same time, *weightRight* values for the left-sided sensors should be smaller or negative, to turn right. The same happens with the sensors from the right side, with the caveat that, on that side, the *weightRight* values should be the ones with positive values. According to this logic, if the sum of the left sensor readings (denoted by $(\text{prox}_1, \text{prox}_2, \text{prox}_3)$) is greater than zero, then (*weightLeft1*, *weightLeft2*, *weightLeft3*) values will increase. Inspired by some well-known approaches [60, 61], below is the mathematical representation for the obstacle avoidance fitness function:

$$F_1 = \sum_{i=1}^{3} (weightLeft_i \cdot \text{prox}_i - weightRight_i \cdot \text{prox}_i) + \sum_{i=4}^{6} (weightRight_i \cdot \text{prox}_i - weightLeft_i \cdot \text{prox}_i). \quad (14)$$

In this representation, F_1 will reward weights that produce the correct turning behaviour, based on sensor inputs. A set of weights that makes the robot avoid an obstacle will produce a positive value of F_1 whether the robot will turn right or left, depending on the triggered sensors. Also, our testing phase turned out that the algorithm is capable of producing good weights values in the particular case when obstacles are situated on both sides. In this case, sensor data from the right as well as from the left side will be retrieved, but the computing principle introduced by the obstacle avoidance fitness function remains the same.

Smoothness is another important objective that should be covered by the robot controller. As the robot controller will call the genetic algorithm at each simulation step when a non-zero sensor value is retrieved, it is indicated to have consistency between the generated values, as avoiding a single obstacle will take several simulation steps. Our second fitness function sums the difference between a proposed individual and the previous selected weights. The objective is to minimise the returned value to avoid sudden big differences in weights, as they are correlated with instant changes in the robot behaviour.

$$F_2 = \sum_{i=1}^{6} \Big(|weightLeft_i - prev_weightLeft_i| + |weightRight_i - prev_weightRight_i| \Big). \quad (15)$$

As can be observed from the above mathematical representation of this second fitness function, F_2 selects each weight value from the individual and determines the absolute difference between it and the previous weight values.

With a brief overview of the implementation established, we can turn our attention to the selection process of a solution, according to the above-introduced functions. The algorithm selection logic is based on the evaluation made by F_1 and F_2 for each individual within a population. To facilitate an appropriate placement into Pareto front, all the objectives were minimised. To achieve that, we had to negate the values returned by F_1. As stated in Sect. 3.3, a solution will be selected if it is on a better Pareto front than others, which in our case means lower or equal values for both the objectives or has a higher diversity, outlined by the crowding distance, if on the same front.

In the proposed controller, obstacle avoidance is achieved through the continuous adaptation of sensor–motor weights, making it the most safety-critical and behaviour-defining objective. Obstacle avoidance and smoothness were selected as the primary optimisation objectives because they directly govern the core interaction between the robot and its environment at the control level, which represents the focus of the present work. Together, they form a complementary and computationally tractable objective pair that supports real-time control, which would be more difficult to achieve with higher-level objectives.

Other objectives, such as energy efficiency or path quality may be considered in future development, but were not selected as primary objectives, as the proposed controller operates at a reactive control level, rather than at a global planning level. The chosen objectives, therefore, reflect a deliberate design trade-off: prioritising real-time reactivity, physical stability and local interaction quality.

From a different perspective, it is known that NSGA-II ranks solutions based on Pareto dominance. When the number of objectives increases beyond three [62], several individuals may become mutually non-dominated, affecting the ability of the algorithm to clearly distinguish between better and worse solutions and promote improvements over generations. Furthermore, the crowding distance becomes unreliable in high-dimensional objective spaces and the required population size increases, imposing a significant memory overhead, as highlighted in Fig. 2. These effects make many-objective NSGA-II computationally impractical for real-time robotic control. Therefore, restricting the optimisation to two well-chosen objectives ensures both effective behavioural evolution and real-time feasibility.

4.4 Interaction with robot controller

With the perspective on how our NSGA-II algorithm works, we can proceed with the explanation of the algorithm integration along with the robot controller. First of all, we used as a starting point the robot controller from our previous research, presented in [58]. In short, the main loop of the controller took the values received from the proximity sensors and assigned them to variables from the sensor membrane inside the enzymatic numerical P system. On the other hand, after each simulation step, the ENPS also had the mission to send the updated values for *leftMotor, rightMotor*, the actual speeds of each wheel. This cycle was designed to continue until the test execution is over (the educational robot reaches the end of the road or fails the test and leaves the designated area).

One can notice that the main loop and the enzymatic numerical P system, essential components in determining the robot behaviour during the tests, were interconnected, exchanging information at each simulation step. As the multi-objective optimisation algorithm used for weight generation requires real time information during the simulation, our new

controller extends the logic of the initial one, by adding some necessary interactions with the NSGA-II algorithm.

First of all, before entering the main loop that will be executed every timestep, the algorithm is initialised, along with the utility class setting up the algorithm parameters introduced above in this section. Then, a first population of individuals is randomly generated. Once the loop will begin, the algorithm will start to improve the population using real time data collected from each proximity sensor.

The main loop is responsible with coordinating the sensor reading, as well as the interaction with the ENPS and NSGA-II which provides the optimisation logic. Of equal importance is that inside the main loop the robot speed will be dictated by setting the *leftMotor* and *rightMotor* values. As the adaptation to the environment challenges must be continuous, the weights are improved while navigating. Next, we introduce a pseudocode representation with the main components of the controller script:

Algorithm 2 Main components of the robot controller

```
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
```

Algorithm 2 emphasizes the mechanism by which the weights are evolved dynamically to optimise them according to the above-mentioned objectives. During each simulation step, the algorithm is run only once. This can be argued as one simulation step is sufficiently small to construct a gradual evolution of weights. By observing the dynamics of a simulation, we can note that in the presence of an obstacle, the sensor values will be constant for a number of timesteps. At each step, the current population is evaluated according to the established objectives and the best individual is selected after obtaining the Pareto fronts and computing the crowding distance. The best individual represents the set of weights that are assigned to the enzymatic numerical P system. After that, a new population is created using classical GA operations. Then, the process starting with evaluating and assigning a new set of weights is repeated, until the end of the simulation.

By analysing this way of working, one can notice that the computational performance might be impacted by the repetitive calls of the NSGA-II algorithm. Costly operations, such as fitness evaluation, sorting, mutation or crossover, should be done when there is a need in having accurate weight values. For example, in the case where no obstacle is detected, all the sensors will have values of zero, so the weight values will become irrelevant. Thus, we constructed and tested some improvements in the way that the genetic algorithm is used inside the main loop of the controller.

The first strategy is related to the aforementioned scenario and the algorithm is called inside the main loop just when an obstacle is detected (i.e. minimum one sensor has a non-zero value). In this way, the simulation speed will increase when no obstacle is reported. Another strategy is to call the genetic algorithm only when at least one sensor changes its value. This strategy comes with a computational improvement during the obstacle avoidance and redirection process, as the controller has less operations to do, but also with a slowdown in the evolutionary learning process.

Before getting into more details about the experimental stage, we mention that our experiments revealed good performance with both approaches, with an advantage on time and memory for the second one, as the number of operations decreased. The results of the comparative analysis will be reported in Sect. 5.

4.5 Development methodology

As the scope of our approach is to improve the robot behaviour by developing a new controller that integrates a multi-objective optimisation algorithm to dynamically generate sensor weights, we used a methodology that enables us to closely observe how the implementation influences the navigation during several simulations.

The components of the methodology are the same as in our previous article [5], namely the following: PeP simulation tool [11], Webots [22], AmbieGen [19] and Python scripts for the robot controller and interaction between the controller, models and simulator. In addition, the NSGA-II algorithm is written as a Python project, having some components called inside the robot controller, as explained earlier. The controller was implemented in Python 3.11 and the entire development process was hosted on a computer with Windows 10 operating system, Intel Core i7-8565U CPU, Intel UHD Graphics 620 (128 MB) graphics card and 16 GB installed RAM.

To some extent, the actual development process still adheres to a *test-driven way of working* [63], as the test sets guide the development process. Moreover, appropriate parameters for the genetic algorithm are typically found by repetitive experimental runs.

The validation of the proposed controller was conducted within the Webots simulation environment, which constitutes a rigorous experimental framework for the objectives of the present study. Webots provides a physics engine with accurate kinematic, dynamic modeling and sensor simulation

incorporating realistic noise characteristics. Building upon our earlier research, we consider that Webots simulation tool provides sufficient resources for our actual scope. As will be outlined in the next section, we were able to construct challenging circular arenas of different radius and with varying number of obstacles to provide reliable experimental results.

The transition from simulation to a potential physical hardware implementation presents some anticipated challenges that are taken into consideration for future development. Aspects, such as communication latencies, between sensors and processor, potential noise or timing jitter are considered. Another factor involves the computational constraints of embedded platforms, which are standard in mobile robotic applications. Because the ENPS model entails the concurrent evolution of several compartments and the NSGA-II algorithm involves evaluating multiple candidate solutions, applying non-dominated sorting, and computing crowding distances, the resulting computational load becomes substantial. A similar approach of moving from simulation, where computational resources are abundant, to physical platforms where efficiency becomes critical, is presented by Buiu and Florea in [64], where the potential of membrane computing in robot control is explored and some limitations are outlined.

The enzymatic numerical P system model, which provides the foundation of the controller is executed with PeP, a dedicated numerical P system tool that allowed us to define the structure for each model instance. The ENPS receives the sensors values as well as the generated weights from the controller and sends back to the controller the computed left and right motor speeds. This is achieved through Python scripts. The clean syntax and the strong support for scientific computing provided by Python were ideal for implementing the evolutionary logic of our multi-objective algorithm that is also interconnected with the robot controller script.

A comprehensive test suite is essential not only for the development process, but also for ensuring proper validation of the proposed solution. In addition to the previously mentioned arenas with obstacles, a variety of roads tests with different sizes and shapes were generated using AmbieGen [65], an open-source tool based on evolutionary search methods.

The present development methodology entails the interdependent use of all these components, as the robot controller gets the information retrieved by running the ENPS in PeP, calls the essential functions of the genetic algorithm and is included in Webots, where the simulation based on both manually created and AmbieGen generated tests is done.

Figure 3 illustrates the proposed flow of obtaining the wheel velocities that are used inside the robot controller using the multi-objective evolutionary optimisation process. It can be observed that the proximity sensor readings represent the input that is fed to the NSGA-II algorithm. The above introduced fitness values guide the optimisation process, which evolves a population of candidate weight sets and selects, at each control step, the most suitable solution. The resulting weights are then injected into the Enzymatic Numerical P system (ENPS) model instance, where the membrane programs compute the wheel velocities of the robot which are then assigned to their corresponding variables, as presented in Algorithm 2. As the controller is directly integrated with the virtual representation of the robot in Webots, the resulting motion influences subsequent sensor readings, contributing to a continuous *perception–decision–action* feedback loop that enables real-time adaptation to dynamic environments.

5 Experimental results

The new controller variant requires evaluation against prior configurations to confirm performance improvements across different operational metrics. PROTOS [66] from Webots simulation tool are used to define test scenarios structured across a range of complexity levels. The current version of the developed controller is expected to prove adaptability, as the set of weights is changed during the simulation according to the triggers sent by the robot in the form of proximity sensor values. In an ideal scenario, the robot should move fast, but still be able to detect and avoid any obstacle. As observed from Eqs. 9 and 10, the motor speeds are directly influenced by weights, so a set of inappropriate weights can cause a constant speed decrease.

When developing an evolutionary algorithm, an indicator of optimisation success may be the convergence of the population. As evolution proceeds through selection, crossover and mutation, inferior solutions are gradually eliminated while higher-quality solutions become dominant. When convergence occurs, successive generations exhibit diminishing variation and increasingly similar fitness values, indicating that the algorithm has identified a stable solution or a set of trade-off-optimal solutions.

In real-time and adaptive robotic control, however, convergence must be interpreted differently. When the optimisation problem itself changes over time due to varying sensor inputs and environmental interactions, full convergence to a static solution is neither expected nor necessarily desirable. Instead, the objective shifts from achieving terminal convergence to maintaining controlled adaptation, where the population remains in a continually responsive state while avoiding excessive instability. In this way, one can observe that in our approach the weights do not converge to fixed values, due to the fact that the NSGA-II is iteratively used inside the main loop, as presented in Algorithm 2. Moreover, fitness evaluation depends directly on continuously changing sensor inputs, as the environment remains non-stationary.

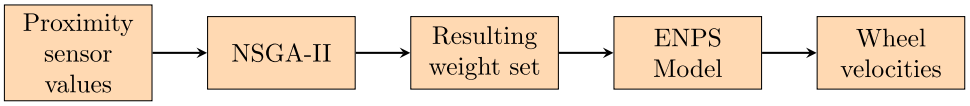


Fig. 3 Information flow within the proposed controller

As stated, collision avoidance is the primary objective when developing such types of controllers. While the purpose of the multi-objective algorithm is both efficiency and performance, it does not explicitly quantify how close the robot operates to surrounding obstacles during execution. In cluttered or narrow environments, trajectories with similar completion times may differ significantly in terms of safety margin. To capture this aspect, we introduce a *minimum positive sensor reading* metric, defined as the minimum positive sensor value retrieved during the simulation. This evaluation metric uses the sensor values that are retrieved during the simulation and is particularly relevant for optimisation approaches, where different solutions may trade efficiency for risk. Moreover, only positive values are considered, as Webots associates the absence of an obstacle with values of zero.

Let $\text{prox}_i(t), 1 \leq i \leq n$ be the value returned by each proximity sensor at timestep $t \in \{1, 2 \ldots T\}$, where T represents the total length of the simulation. Then, the minimum positive sensor reading, referred for simplicity as obstacle clearance $C_{\min}$ is defined as:

$$C_{\min} = \min\{\text{prox}_i(t) \mid 1 \leq i \leq 6,\ t \in [1, T],\ \text{prox}_i(t) > 0\}. \tag{16}$$

The above formula takes the minimum positive sensor reading at each simulation step, indicating the closest approach of the robot to any obstacle. By keeping this as an evaluation metric, the ability of the multi-objective algorithm to explore efficient trajectories while maintaining the safety of each solution is quantitatively assessed.

The inclusion of a *smoothness* objective limits abrupt variations between successive weight solutions, promoting gradual adaptation rather than instability. During our experimental phase, we observed that local stabilization occurs when the robot moves in a uniform situation (e.g., straight corridor, no obstacles nearby), whilst in scenarios with obstacles in proximity, the solutions fluctuate within an acceptable range, meaning that the robot exhibits a natural behaviour.

To complement the smoothness objective, we introduce an additional indicator based on the temporal variation of wheel velocities. Even if this metric is not included as an optimisation objective in the algorithm to avoid over-constraining the evolutionary search, it is intended for evaluation. Let *leftSpeed(t)* and *rightSpeed(t)* be the motor speed values at the discrete time step t, for a trajectory of length T. Then, the velocity variation Δv can be defined as follows:

$$\Delta v = \frac{1}{T-1} \sum_{t=1}^{T-1} (|leftSpeed(t+1) - leftSpeed(t)| + |rightSpeed(t+1) - rightSpeed(t)|). \tag{17}$$

According to the above formula, lower values of Δv indicate smoother behaviour, corresponding to smaller changes in wheel velocities from consecutive steps. The metric is used to detect short-term control oscillations that may not be explicitly penalized by the smoothness objective, F_2 that is related to the weight values.

To outline both the advantages and potential disadvantages of the new robot controller, we conducted the whole testing phase as a comparative analysis between the NSGA-II controller and the previous ones. From our previous work, we selected just the models with the most satisfying results in passing the tests. These are Π_{M_2} (Basic Model with Rotation), Π_{M_3} (Refined Model) and Π_{M_5} (Linear Regression-Based Model). Thus, we compared the NSGA-II controller running Π_{M_2}, the previous classical controller running Π_{M_3} with empirical weights and the Linear Regression-Based Model, Π_{M_5}. The Refined Model, Π_{M_3}, also worked fine with dynamically assigned weights, outperforming the static weights. However, due to the simplicity of Π_{M_2} structure, in the below presented experiments we used Π_{M_2} as the backbone formal model for the NSGA-II controller.

Complementing the descriptive and statistical analysis that will be presented, Fig. 4 illustrates three road tests with the overlaid trajectories associated with the three controller variants that are considered in the comparative analysis of this study, where the path of the NSGA-II controller is highlighted in green, the linear regression-based (using Π_{M_5}) in red and the controller using empirical weights and model Π_{M_3} in yellow.

Previous studies showed that the static controllers demonstrated good behaviour in terms of safeness and smoothness. However, these results relied on a laborious process, which involved collecting data and applying linear regression or "trial and error" for finding empirical values. Apart from this time-consuming process, the major disadvantage of static weights is the lack of adaptability across different scenarios. As a result, a controller tuned for one scenario may perform poorly in others. Unlike the static weighted approach, NSGA-II explicitly maintains a population of solutions that approximate the Pareto front. This enables the exploration of diverse trade-offs between competing objectives, thereby

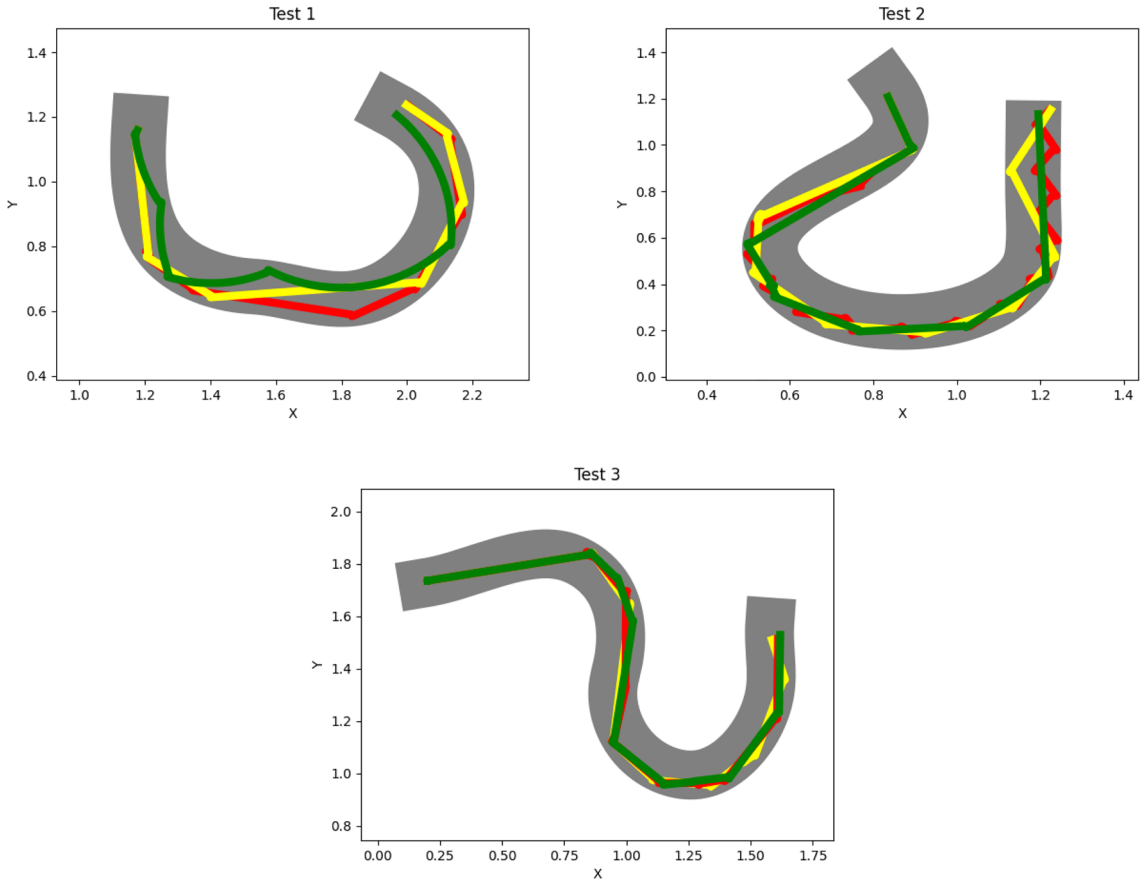


Fig. 4 Trajectories of different controller variants

Table 1 Obstacle clearance as a safety indicator

Road test number	NSGA-II	Linear regression	Empirical weights
Test 1	0.0485 ± 0.001	0.0390	0.0480
Test 2	0.0493 ± 0.002	0.0299	0.0490
Test 3	0.0490 ± 0.001	0.0419	0.0489

Table 2 Average speed variation as a smoothness indicator

Road test number	NSGA-II	Linear regression	Empirical weights
Test 1	0.0402 ± 0.007	0.0923	0.047
Test 2	0.0644 ± 0.006	0.1082	0.067
Test 3	0.0301 ± 0.007	0.0741	0.044

avoiding fixed scaling. Consequently, NSGA-II provides greater robustness to environmental variability.

Tables 1 and 2 report the evaluation results for the NSGA-II controller that uses dynamic weights and the two variants of controllers that use static weights (regression-based and empirically chosen) that are appropriate for this type of roads. Even though the static weight controllers were specifically developed for road tests scenarios, the results are closely aligned; moreover, the NSGA-II controller exhibits a slight advantage, indicating an effective selection of solutions achieved through the continuous optimisation during the test execution. However, we recall that this study aims to provide an adaptive mechanism for balancing multiple navigation objectives while maintaining safe operation. In this regard, substantial gains in safety metrics (such as obstacle clearance) were not necessarily expected, since the previous controller variants were already proven to be secure. From this perspective, the closely comparable clearance results relative to past versions indicate that the improvements in efficiency are achieved without compromising the opera-

tional safety. While these metrics were not directly employed to motivate the proposed approach, they offered essential insights into critical system parameters. Preserving these core characteristics represented a priority during the design of the NSGA-II controller. Consequently, this foundation facilitated a seamless transition to a comprehensive performance analysis of the controllers, which is presented in detail below.

5.1 Performance metrics

This testing phase was composed of two different categories of scenarios: *road tests* and *arenas with obstacles*. Starting with the road tests, we had five sets of roads, each with different shape and length. As the roads were generated using AmbieGen, we obtained routes characterised by certain properties through the modification of parameters for road generation inside the open-source code of the tool. In this way, we obtained five sets of roads with incremental values for the minimum distance of a generated road, maximum distance to go straight during a road and the map size.

Accordingly, five sets of 4 roads were obtained, as follows:

- Set 1 : (min_len : 5, max_len : 15, map_size : 150)
- Set 2 : (min_len : 10, max_len : 30, map_size : 200)
- Set 3 : (min_len : 15, max_len : 40, map_size : 400)
- Set 4 : (min_len : 20, max_len : 60, map_size : 500)
- Set 5 : (min_len : 100, max_len : 150, map_size : 1000).

As the dynamic assignment of weights introduces an element of non-determinism in the robot behaviour, we ran each test with the controller using NSGA-II for 5 times. This allowed us to better observe convergence tendencies of the dynamic weights and assess the robustness of the adaptive mechanism. In this way, a representative performance can be reported, rather than a single potentially biased execution. The actual version of controller (referred for simplicity during the experiments as *NSGA-II controller*) managed to pass all the experiments. We collected relevant data from each simulation and computed the average value for each metric monitored throughout our experiments.

In contrast, the controllers based on static weights (empirically chosen or linear regression based) are fully deterministic. By running several runs, it can be noticed that given identical initial conditions and using static weights, the same sensory inputs will produce the same control actions at each simulation step, making repeated executions experimentally redundant. Consequently, our experiments highlighted that the robot follows an identical trajectory in every execution on the same test scenario. Therefore, a single experimental run will illustrate the behaviour and performance of the static-weight controllers.

Let us introduce the performance metrics followed in our initiative. Given the observation that all the participating controllers passed all the test scenarios, the first criterion recorded was the *simulation time*, i.e. the time needed to reach the end of the road. To analyse the contribution of dynamic weights to the efficiency of the path followed by the robot to pass a test, the second criterion was the *distance* travelled during the simulation. Webots provides the values expressed in radians for the left and right encoders, measuring the rotation of the wheels. By having access to the wheel radius, we were able to compute the linear distance travelled by each wheel and then obtaining the average distance expressed in millimeters. Another quantitative parameter considered throughout the experiments was the *memory* consumed during the simulation. As the NSGA-II controller performs more sophisticated operations, it was anticipated that the memory consumption will increase accordingly. However, to reduce the impact of this aspect, we opted for the use of Π_{M_2} along with the NSGA-II controller, as it has a simpler structure with fewer membranes and programs, as highlighted in [5].

As mentioned earlier, since the dynamic weight adaptation introduces variability, we ran multiple times each road test and computed the *mean* $\pm$ *standard deviation*, as presented in Tables 3 and 4. As can be seen, the dynamic weight adaptation based on NSGA-II demonstrates stability despite its adaptive nature. The Coefficient of Variation (CV) for the execution time remains within the acceptable range for metaheuristic algorithms (CV: 4.0–9.3%). The results for distance exhibit even lower variation, having a CV between 1.1 and 5.9%. Notably, larger problem instances, such as Sets 4–5, maintain comparable or lower relative variance than the smaller instances, suggesting that the adaptation mechanism scales appropriately with problem complexity. For example, despite having the longest paths (mean: 7229.1), Set 5 exhibits the lowest CV (1.1%), demonstrating that the increased problem size does not compromise solution consistency.

The observed stability can be attributed to the calibration of NSGA-2 parameters, obtained after several experimental runs, as stated in Sect. 4.2. By this means:

- The population size influences the level of genetic diversity available for exploring the solution space while balancing computational efficiency and search stability.
- Tournament selection prevents premature convergence while ensuring consistent progression toward high-quality regions.
- A moderate mutation strength allows weight adaptation without dramatic fluctuations.
- Uniform crossover with 50% contribution from each parent balances exploration and exploitation, preventing offspring from deviating too far from proven solutions.

All the above-mentioned aspects should be considered when discussing the consistency and optimisation reliability of evolutionary computation.

Table 3 Robot controllers performance by time (s)

Test type	NSGA-II	Linear regression	Empirical weights
Set 1	**69 ± 5.7**	116.5	116.75
Set 2	**57.15 ± 4.3**	99.5	81.25
Set 3	**113.1 ± 4.5**	203.75	156
Set 4	**103.3 ± 9.5**	178.5	141.5
Set 5	**222 ± 12.7**	379.2	363.4

Bold values denote the best result obtained for each set

Table 4 Robot controllers performance by distance (mm)

Test type	NSGA-II	Linear regression	Empirical weights
Set 1	**2631.7 ± 56.43**	2920.75	2830.5
Set 2	**2556.1 ± 32.9**	2873	2704
Set 3	**4441.5 ± 55.8**	4443.25	4467.25
Set 4	**4128.4 ± 77.6**	4166.5	4284
Set 5	**7240 ± 82.1**	7591	7627

Bold values denote the best result obtained for each set

Table 5 Robot controllers performance by memory (bytes)

Test type	NSGA-II	Linear regression	Empirical weights
Set 1	1,510,710.2	761,736	**102,400**
Set 2	1,761,713.7	**1,465,050**	1,966,080
Set 3	695,910.4	587,837	**116,293**
Set 4	2,236,705.8	**105,700.5**	123,149
Set 5	2,424,012.8	169,264	**139,264**

Bold values denote the best result obtained for each set

One can notice that the NSGA-II controller has better performance with respect to time (Table 3) and distance (Table 4) than the other two controllers included in the experiments, whereas in terms of memory (Table 5), it ranks lowest—a predictable consequence due to the higher complexity of the algorithms involved. The significant advantage in terms of time may be explained by the right choice of the weight values that was done by the multi-objective algorithm. In this way, the robot "knows" better how to escape easily from the moments where an obstacle is nearby. For example, a set of small weights will result in short rotations. Even if the robot will finally avoid the obstacle, the time required to complete this mission is considerable. Moreover, even if the static weights offer safety in terms of passing the tests, a multi-objective optimisation approach comes with the notable advantage of searching for appropriate weight values during the entire simulation. Over time, the controller discovers new strategies for different challenges, adapting the solutions based on the success or failure of the previous actions. This is also the main cause of having a non-deterministic behaviour.

Regarding the second indicator, that is the total distance traveled to reach the goal, we initiate some observations by examining the former approach. Static weights may often result in small rotations or wide avoidance due to the fact that a single scenario typically contains different types of curves, positioned randomly and the weights have constant values. This will increase the distance traveled, as the robot proximity sensors might be triggered more often. The selection process of the multi-objective algorithm will better reward individuals that produce smooth movement, as well as performant obstacle avoidance.

To better assess the statistical significance of the performance differences, we employed a nonparametric statistical test, namely the Wilcoxon signed-rank test [67]. We used a one-sample Wilcoxon signed-rank test against deterministic baselines, comparing a larger number of NSGA-II controller runs ($n = 20$) against the baseline deterministic values obtained by Π_{M_3} and Π_{M5}. With respect to this type of nonparametric test, it is important to specify that the multiple NSGA-II controller runs on the same road were treated as repeated measurements on the same experimental unit (the road) and were used solely to estimate the central tendency of this controller performance. For this type of test, we decided to increase the sample size compared to the previous experiments to better asses the variability. Thus, for each road and metric (time, distance, memory), the median performance of NSGA was computed and compared against the corresponding baseline value using a one-sample Wilcoxon signed-rank test. As mentioned earlier, each set consists of four distinct roads with similar characteristics. As the roads within a set are not identical, exhibiting differences in terms of structure and length, the statistical tests were not applied to pooled measurements across roads. Instead, the results were interpreted at the set level by assessing consistency of outcomes across roads, rather than by aggregating raw measurements or p-values.

With respect to execution time, the NSGA-II controller achieved the minimum $p = 0.0625$ attainable for this number of observations, indicating significant superiority. Regarding the total distance, the comparison between the NSGA-II controller and the static controllers revealed lower distances for the dynamic controller in every case. In terms of execution time, the dynamic controller significantly outperforms the static controller, the distance analysis also showed that NSGA-II consistently produces shorter routes, even if the differences are comparable. As shown by the previous results, the NSGA-II controller required significantly more memory than both baselines ($p < 0.001$ for both comparisons). These results confirm the computational overhead inherent to population-based evolutionary algorithms.

To summarize, the Wilcoxon signed-rank test demonstrates that the NSGA-II controller provides statistically significant and consistent improvements in execution time across different categories of road tests, while clear improvements in distance are more dependent on road characteristics. As expected, the dynamic weight assignment exhibits higher memory usage than the static approaches, as the continuous optimisation process requires multiple candidate solutions simultaneously. The statistical methodology employed provides an additional demonstration that the above-presented performance differences reflect true algorithmic improvements rather than random chance.

The second testing scenario was based on a circular arena filled with cubical obstacles of different dimensions (Fig. 5). With the aid of this experimental setup, our objective was to analyse the robot behaviour in a scenario that presented notable differences from the previous one, having narrower spaces and a greater density of obstacles. In addition, this experimental stage was more extensive than the first one with road tests, as we varied both the arena radius and the number of obstacles. We used the following values:

- *radius (m): 0.8, 1, 1.2, 1.4, 1.6*
- *number of obstacles: 5, 7, 9, 11, 13.*

Thereby, we obtained 25 different arenas. For each of them, we performed 5 runs with the NSGA-II controller, one run using the controller with the Linear Regression-Based Model and another with empirical weights. We then computed the mean and the standard deviation of the runs carried out with the NSGA-II controller.

Remark 4 As this paper includes only a subset of results, the entire set of experiments can be consulted at [68].

Figure 6 illustrates the plots with the results for each of the three metrics when the radius of the arena is 1.2 m. We can observe that the NSGA-II controller maintains its superiority in terms of time; however, the differences are no longer as

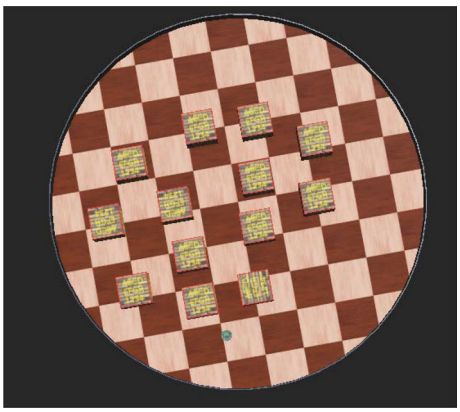


Fig. 5 Circular arena of radius 1.2 and 13 obstacles

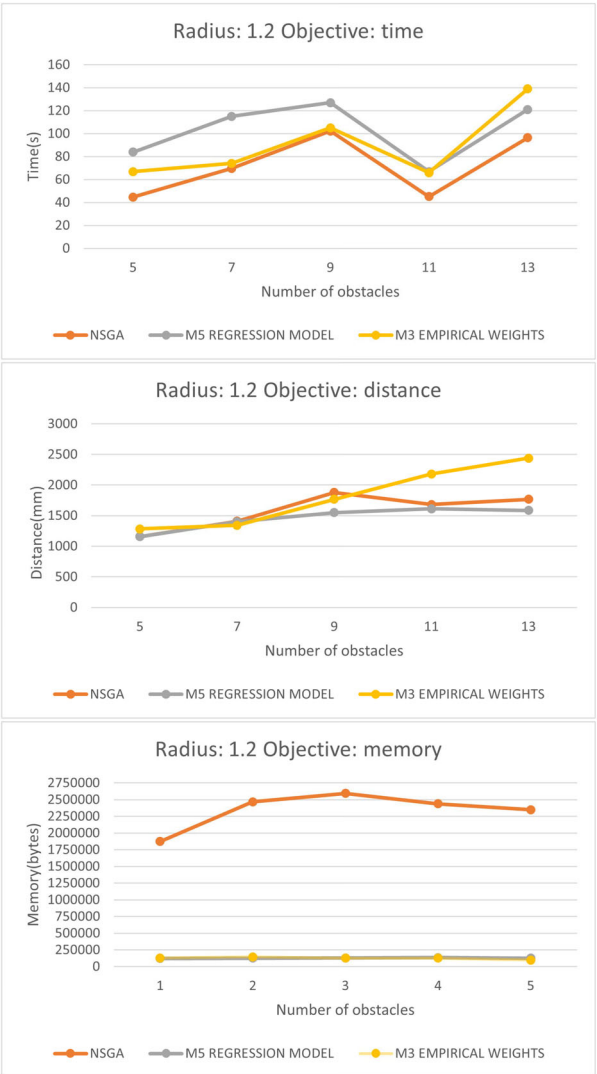


Fig. 6 Simulation results for circular arena of radius 1.2

significant. Moreover, for some number of obstacles, the controllers using static weights obtained better distance results. This can be explained by the fact that we consider a test to be passed when the obstacle departs from the obstacle-filled area, via any side. This aspect further emphasizes the non-deterministic character of the NSGA-II controller, as the exit point changed from one simulation to another. Nevertheless, even if the NSGA-II controller tended to follow longer paths than the previous version of controllers, the avoidance was performed faster, leading to favorable execution times.

A closer look at the memory usage metrics underlines the major differences between the NSGA-II controller and the previous ones. As for the controllers using Π_{M_5} and Π_{M_3}, the logic of generating weights is decoupled from the controller which only takes the static values (empirical and regression-based) from a separate file, it was expected that in terms of memory usage, the experimental outcomes of the two controllers would align. In contrast, it can be observed that the NSGA-II controller obtained notably higher values. The significant difference is explained by the additional functionality introduced for the dynamic weight generation. As stated in Sect. 2, the multi-objective algorithm is initiated and populations are continuously improved inside the main loop of the controller, ensuring the dynamic generation of weights during robot navigation. Building upon the prior comparison with the previous two controllers, where the main loop was principally handling the communication with the enzymatic numerical P systems and assigning velocities, it can be observed that in the case of the NSGA-II controller the weights are obtained by running the costly tasks of the multi-objective algorithm for one generation and then updating the actual weight values with the best individuals. As presented in Sect. 3.3, several functions operate in conjunction to create, select and evaluate individuals. As a result, each call of these functions implemented in Python increases the memory demand. A common strategy used when reducing the memory footprint of Genetic Algorithms is to scale the population size, employing a reduced size and storing only the current population and immediate offspring. As presented earlier, our analysis demonstrated that a population of 50 individuals retains the adaptive characteristics of the algorithm while lowering memory usage significantly, as stated in Fig. 2. Nevertheless, aspects, such as lightweight encoding of individuals and parameters (for example, using fixed-point integers, similar to the original variant of static controller), preserving an elitist NSGA-II implementation (which limits memory usage to storing only the current population and offspring at each generation, avoiding any external solution archives) are considered in our approach to reduce RAM requirements while preserving the real-time evaluation dynamics that guide evolutionary adaptation. As a final remark concerning memory usage analysis, the experiments have shown a correlation between a decreased number of

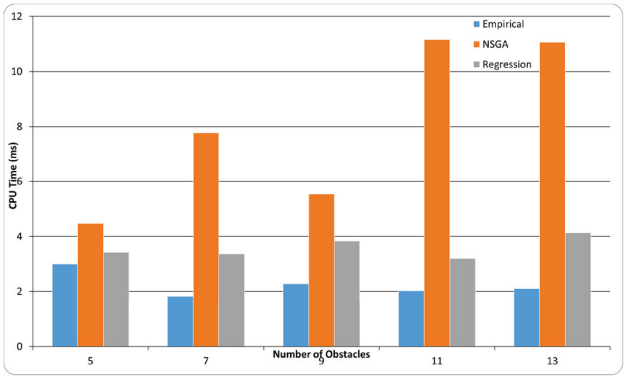


Fig. 7 Average CPU time per step by controller inside the circular arena of radius 1.2

operations and memory usage; however, in the absence of a prior analysis, the navigation accuracy may be compromised.

While memory consumption constitutes a relevant indicator of computational performance, CPU time per simulation step represents a complementary key indicator in online simulation scenarios. Figure 7 highlights a comparative analysis of the average CPU time per step achieved by the three controller variants inside the circular arena of radius 1.2. The analysis of the static controllers (regression-based and empirical weights) reveals comparable results, with a slight advantage for the controller using regression-based weights and model Π_{M_5}. However, the existing difference can be explained by the structure of the enzymatic numerical P system model Π_{M_5} that has only 28 variables and 23 programs, performing simpler arithmetic operations. In comparison, the model of the controller using empirical weights, Π_{M_3}, has a total of 54 variables and 52 programs.

As anticipated, upon considering the dynamic NSGA-II controller, a significant difference is observed, reaching a maximum of 11.1559 ms. In line with the previous experiments, for this type of non-deterministic controller we included the mean value of five simulations on each test case. As expected, the dynamic controller exhibits higher computational cost per step due to its multi-objective optimisation approach. During the experiments, it was observed that the average cost increased when the robot was avoiding an obstacle. In the remaining periods, when the robot moved straight, having equal wheel velocities, the values were similar to those recorded for the static variants. This can be explained by the strategy of integrating the NSGA-II algorithm into the main logic of the controller, which requires at least one proximity sensor to be triggered to call the algorithm. Moreover, in such situations, the algorithm will evaluate multiple candidate solutions and perform the non-dominated sorting, increasing the computational complexity.

As mentioned in Sect. 4.4, multiple strategies of incorporating the Genetic Algorithm (GA) into the controller's main

logic were studied. We recall each strategy that was evaluated:

1. Periodic: Invocation of GA at each simulation step.
2. Obstacle-triggered: Invocation triggered by obstacle detection (i.e. at least one proximity sensor value reports a value).
3. Change-triggered: Invocation triggered by a change in the value of at least one proximity sensor.

The "*Periodic*" strategy implies significant computational overhead, due to the fact that the optimisation process takes place at every simulation step, regardless of whether the robot is moving straight, as no obstacle is detected. Thus, as the optimisation process is guided by the sensor values, a massive number of redundant calls are performed during the simulation. As presented in Sect. 4.1, the motor speeds are equal and constant when no obstacle is detected, regardless of the weight values. Taking this into account, optimising the weight values when no obstacles are detected and provides no benefit. To mitigate the computational overhead, an alternative strategy that limits the number of redundant invocations was developed. The "*Obstacle-triggered*" strategy implies that the optimisation will take place only when at least one sensor value reports the presence of an obstacle (i.e. returns a positive value). This strategy lets the robot navigate with constant weight values in obstacle-free regions, leading to similar computational statistics as the static controllers when no obstacle is detected. This strategy enables a reactive behaviour, as from the first step where an obstacle is detected the Genetic Algorithm is called and the weight values are scaled according to that particular situation.

The traces extracted from Webots revealed that a simulation step constitutes a minimal measurement unit, which is visually imperceptible during obstacle avoidance maneuvers and the sensor readings remain constant over multiple consecutive steps. Thus, a third strategy has been obtained, namely the "*Change-triggered*". This strategy implies that the Genetic Algorithm is called only when at least one proximity sensor changes its value from the previous simulation step. As presented in Algorithm 2, the evolutionary algorithm is run for one generation during one simulation step. In this way, the use of the "*Change-triggered*" strategy will keep the adaptability in dynamic environments, but will come with the disadvantage of limiting the evolutionary process.

Table 6 highlights the main differences between the GA integration strategies presented above. The results represent the average values that were collected from 5 test cases of different complexities: circular arenas of radius 1.2 and variable number of obstacles (5, 7, 9, 11 and 13). One can observe that the second and third strategies achieved similar results, both being suitable for the type of scenarios of the current study. The most significant difference between the three strategies arises from the number of algorithm calls. As the algorithm is invoked at each iteration in the *Periodic* approach, the number of calls will be equal to the number of simulation steps. The other strategies impose a triggering logic that governs the execution of the Genetic Algorithm, thereby yielding fewer invocations, according to the specified constraint. Moreover, the increased number of algorithm calls leads to higher execution time, since the execution of each iteration includes the necessary operations to execute one generation of NSGA-II. This execution mechanism also influences the reported memory usage, which increases with the number of invocations.

To some extent, the above comparison replicates the components of an ablation study, a commonly used procedure, especially in Artificial Intelligence, where certain elements of a system are removed in order to achieve a better understanding of the functionality [69].

Figure 8 highlights the individual time performance for each controller. Starting with the NSGA-II controller, which is introduced in this paper, one can observe that the highest duration reached across the simulations is located near the value of 130. Extending the comparison with the alternative models, the second chart reveals the presence of three values exceeding the threshold of 150 for the controller running the Π_{M_5} regression model. In addition, the last chart indicates that the controller using Π_{M_3} with empirical weights recorded values above 180 in two experimental runs and achieved a value of nearly 140 in another experiment. Moreover, looking at individual tests, by taking the values for each number of obstacles and radius, one can notice that NSGA controller demonstrates superior performance in terms of time compared to the other two variants.

In Fig. 9, the distance recorded by each model during the experimental phase inside circular arenas having different radius size and number of obstacles is presented. In some sense, we can affirm that adding new obstacles made the simulation shorter in some cases. Despite the fact that it seems unusual, this behaviour can be justified by the fact the addition of a new obstacle can lead to new routes that are potentially more challenging, but shorter than those inside an arena with widely spaced obstacles. In fact, the obstacles were "strategically" positioned following an analysis of the behavioural patterns observed in prior tests with a lower number of obstacles.

Resuming the analysis of the distances presented in the charts, one can observe that NSGA and Π_{M_3} empirical weight controllers obtained similar results, with an overall advantage for the NSGA controller. However, it is worth reiterating that running the NSGA algorithm inside the controller leads to non-determinism, so the values presented in the first chart are the average of 5 simulations and not the best distances obtained.

As a preliminary observation, the experiments we conducted indicated that the dynamic assignment of weights

Table 6 Strategy comparison using different metrics

Strategy	GA calls	Avg. speed variation	Avg. clearance	Execution time	Memory
Periodic	3486 ± 122	0.103 ± 0.022	0.031 ± 0.004	252.64 ± 18.2	19921984
Obstacle-triggered	523 ± 23	0.10 ± 0.034	0.044 ± 0.004	115.2 ± 9.43	11829824
Change-triggered	219 ± 11	0.11 ± 0.011	0.048 ± 0.003	109 ± 9.24	8708096



Fig. 8 Time performance inside circular arenas for each controller



Fig. 9 Distance recorded inside circular arenas for each controller

introduced the exploratory component whilst not compromising the safety, as all the tests were passed. In some sense, we can affirm that the static weights constitute a "safe but slow" alternative, being centered on passing the tests, but with no control in improving some performance metrics in changeable environments.

5.2 Performance consistency under NSGA-II hyperparameter variation

To evaluate the stability of the proposed approach against hyperparameter perturbations, we conducted a systematic sensitivity analysis on three hyperparameters that most heavily influence the balance between convergence and diversity achieved by NSGA-II: the population size P, the mutation strength m and the number of genes mutated per operation g. We adopted a one-at-a-time (OAT) perturbation design [70] whereby each hyperparameter is varied symmetrically around its baseline value (as presented in Sect. 4.2), while all others remain fixed. The OAT design is an appropriate methodology for sensitivity analysis, as it isolates the individual effect of each hyperparameter on performance. For the remaining hyperparameters (tournament size of 3 and uniform crossover), the preliminary checks performed during the algorithm development phase, where both parameters were varied within different ranges (tournament_size $\in 2, 3, 5$, single-point vs. uniform crossover), revealed that no significant qualitative changes in convergence behaviour or final solution quality can be observed, thereby supporting their treatment as fixed in the present analysis.

Similar to the above presented experiments, taking into account the non-deterministic nature of the NSGA-II algorithm, we performed 5 independent test runs for each hyperparameter configuration. The resulting variants were evaluated on the following metrics that were previously introduced: distance, time, average speed variation and obstacle clearance.

Table 7 illustrates the evaluation results for the controller across different NSGA-II hyperparameter configurations, using one of the most challenging testing scenarios: the arena of radius 1.5 containing 13 obstacles. It can be noticed that each configuration produces interpretable patterns that are consistent with the exploration-exploitation logic that was previously discussed in this study. For example, the number of genes mutated per operation (g) yields similar performance for the baseline across all the metrics when equal to one, as the search behaviour remains essentially local and incremental. In contrast, when doubled to 6, it produces a significant degradation, with a considerable increase in distance and time. Moreover, in 2 of the 5 runs, the robot got stuck and the simulation could not continue.

The observed degradation in performance when increasing the number of mutated genes from 3 to 6 may be explained by the online nature of the proposed controller. Since each individual encodes 12 motor weights, mutating six genes modifies half of the controller weights at once. Such large perturbations frequently disrupt the sensor–motor relationship that is fundamental for effective obstacle avoidance, producing offspring that differ substantially from previously successful solutions. As a result, the evolutionary search becomes excessively exploratory, causing unstable steering behaviours, longer trajectories and increased navigation times. As stated, in some cases, the generated weight configurations may compromise the desired balance between left and right avoidance responses, leading to failed runs. In contrast, mutating only 3 genes still allows the controller to progressively refine its behaviour without sacrificing navigation stability. Consequently, the configuration with 3 mutated genes achieves better overall performance and robustness than a more aggressive mutation strategy.

Regarding the mutation strength (m), one can notice that the increased value simultaneously degrades the distance and time, reflecting insufficient refinement of promising candidates, whilst a lower value yields worse values for distance at a near-baseline completion time.

Concerning the population size, the experimental results indicate that a population size of 50 provides the best balance between search diversity and adaptation speed. Smaller populations suffer from reduced exploration capabilities and premature convergence, resulting in longer trajectories and increased navigation time. In contrast, larger populations introduce excessive exploration and slower adaptation to environmental changes, which negatively affects real-time navigation performance. The findings align with the expected behaviour, thus indicating that for the proposed online controller, a moderate population size of 50 offers the most effective trade-off between optimisation quality and responsiveness.

Taken together, these patterns indicate that the resulting baseline represents the optimum within the explored hyperparameter configurations. Notably, small perturbations produce gradual, theoretically anticipated degradations rather than abrupt shifts in the robot behaviour; system failure occurs only under a single, aggressive perturbation ($g = 6$). The variability of the results should not be interpreted as an evidence of fragility, but as confirmation that the algorithm responds to its hyperparameters in an interpretable manner.

6 Discussion

A common practice in the evaluation of autonomous navigation and obstacle avoidance systems is to benchmark the proposed approach against established baseline methods to obtain a more objective assessment of performance and to provide a broader context for interpreting the results. However, given the diversity of existing approaches, a comprehensive comparative study deserves separate, detailed consideration and remains a subject of interest for future work. Nevertheless, to provide an initial reference point, this section presents a brief comparison between the proposed approach and a widely applied method in robotic motion planning, namely Artificial Potential Fields (APF) [71, 72].

The APF method was selected as a representative baseline due to its increasing adoption in the obstacle avoidance literature. This choice is also motivated by its computational efficiency and conceptual elegance, modeling the robot environment as a scalar potential field, where obstacles produce repulsive forces and a specified goal position generates an attractive force [73].

It can be noticed that our proposed controller operates as a reactive obstacle avoidance system with no goal-directed behaviour. Consequently, in obstacle avoidance scenarios where no explicit destination is defined, the attractive component is often replaced by a constant forward-driving force. This adaptation preserves the fundamental APF principle of combining virtual forces while enabling real-time reactive behaviour with minimal computational overhead. In the above presented testing scenarios, the road boundaries, arena edges and obstacles can be modeled as instances that generate repulsive potentials when the robot approaches them. By summing the repulsive forces from nearby obstacles, the controller constantly updates the trajectory to steer the robot away from collisions. The approach is well-suited for embedded robotic systems, as it only requires local sensor feedback, bypassing the need for global maps or

Table 7 Evaluation metrics under NSGA-II hyperparameter perturbations

Configuration	Distance (mm)	Time (s)	Avg. speed variation	Obstacle clearance
Baseline ($P = 50$, $m = 0.5$, $g = 3$)	1704 ± 34.17	168 ± 8.09	0.191 ± 0.015	0.04 ± 0.001
$P = 25$	2181 ± 162.4	203 ± 12.5	0.175 ± 0.010	0.038 ± 0.003
$P = 100$	1963 ± 149.6	225 ± 18.4	0.181 ± 0.001	0.04 ± 0.0007
$m = 0.25$	2278 ± 305	177 ± 32	0.167 ± 0.019	0.042 ± 0.002
$m = 0.75$	2431 ± 231.2	246 ± 32	0.171 ± 0.02	0.041 ± 0.002
$g = 1$	1895 ± 231.4	181 ± 36	0.190 ± 0.009	0.043 ± 0.002
$g = 6$	2976 ± 342[a]	259 ± 28[a]	0.2 ± 0.09[a]	0.04 ± 0.005[a]

[a]Two out of five runs failed to complete, as the robot was unable to navigate past the obstacles in the arena; reported values are computed over the three successful runs

Table 8 Total distance traveled: APF vs. NSGA-II ENPS controller

Road test number	NSGA-II	APF
Test 1	1955 ± 32.22	2000
Test 2	1947 ± 42.39	1982
Test 3	3287 ± 54.11	3425
Test 4	4106 ± 29.72	4122
Test 5	7290 ± 77.47	7434

Table 9 Average obstacle clearance: APF vs. NSGA-II ENPS controller

Road test number	NSGA-II	APF
Test 1	0.0483 ± 0.001	0.0424
Test 2	0.0494 ± 0.001	0.0451
Test 3	0.0422 ± 0.005	0.0389
Test 4	0.0391 ± 0.002	0.0381
Test 5	0.0425 ± 0.004	0.0419

Table 10 Average speed variation: APF vs. NSGA-II ENPS controller

Road test number	NSGA-II	APF
Test 1	0.0602 ± 0.009	0.0604
Test 2	0.0560 ± 0.008	0.0311
Test 3	0.0423 ± 0.009	0.0391
Test 4	0.0394 ± 0.006	0.0319
Test 5	0.0502 ± 0.009	0.0444

computationally expensive path-planning algorithms. As a result, APF controllers often achieve smooth and responsive obstacle avoidance while maintaining low implementation complexity.

Despite the advantages, there are some potential limitations when employing the classical variant of APF in complex obstacle avoidance scenarios. The most significant issue is the occurrence of local minima [74]. Since the robot follows the combined force vector, situations may occur where the forces balance each other. In such scenarios, the robot gets stuck even if a clear path exists. This problem is particularly common is narrow corridors or labyrinth environments, where the geometry of the obstacles creates unfavorable force distributions. Similarly, when repulsive forces from opposite sides become comparable in magnitude, the robot may alternate the steering direction, resulting in inefficient motion and increased travel time

Tables 8, 9 and 10 compare the navigation performance of a reactive APF controller with constant forward attractive bias implemented in Webots and our proposed controller using NSGA-II. Three of the previously introduced metrics are considered: total distance traveled, obstacle clearance and average speed variation. The results were collected after running 5 simulations using the non-deterministic NSGA-II based controller and one simulation using the deterministic APF controller on a selection of 5 road tests with different shape and length extracted from the sets defined in Sect. 5.

The two controllers show similar performance across all the presented metrics, with neither approach gaining a clear or consistent advantage. In terms of total distance traveled, the NSGA-II based controller produces a marginally lower value. However, in some experiments the difference falls within the standard deviation, thus resulting in comparable values. Regarding the obstacle clearance, both the controllers maintain distances within safe limits, with the NSGA-II based controller achieving higher mean values. The results are consistent with the design principles of the compared controllers, as the first objective of the proposed NSGA-II implementation rewards weights that safely avoid obstacles, whilst APF relies on repulsive potential fields to directly penalise proximity.

Finally, the APF controller exhibits slightly lower average speed variation, reflecting the continuous nature of the potential field force computation. Changes in wheel velocities occur gradually as obstacle distances vary, producing smooth steering corrections and reducing abrupt accelerations or decelerations.

Since the two controllers operate on different principles, a direct comparison of their execution time and memory consumption would not yield meaningful results. As the controller proposed in this study uses ENPS to compute the wheel velocities at each simulation step, its software architecture is based on a cellular structure that distributes the computation process across several membranes, each one encapsulating rules and variables that govern the robot behaviour based on sensor inputs. In addition, NSGA-II continuously adjusts the weight values that are passed to the ENPS, enabling the robot to dynamically adapt its navigation strategy to different environments without any human intervention.

Both controllers run within Webots, which provides a physically accurate simulation environment with realistic sensor noise and experimental infrastructure that supports model-based testing and formal verification. However, the use of PeP simulator for executing the ENPS model instance and the integration of the multi-objective algorithm add an expected computational overhead for the proposed controller. In comparison, the capacity of online self-reconfiguration is absent from the APF approach, whose formulas are fixed at the design time. As demonstrated, despite these fundamental architectural differences, the above measured navigation metrics remain comparable. Consequently, the results highlight that the integration of the PeP simulator and the NSGA-II optimisation layer, while introducing additional overhead, provides adaptability, reliability and dynamic tuning of ENPS weight parameters.

7 Conclusions

A robot controller based on Enzymatic Numerical P systems models has been tuned using an advanced multi-objective optimisation algorithm. The use of static sensor weights has been replaced by a method to dynamically assign values in order to achieve good obstacle avoidance performance and smooth movement. The development methodology entails continuous testing of the resulting controller in order to analyse the performance of the multi-objective algorithm. An NSGA-II adaptation with two objectives is used. The resulting implementation is integrated with the controller that interacts with the instance of NSGA-II and the ENPS model during the simulation. The definition of three ENPS models is introduced along with the rationale behind the development of each one, revealed during the evaluation of its predecessor.

A comparative analysis between the NSGA-II based controller and the controllers presented in our previous work has been elaborated, considering multiple relevant evaluation metrics. The simulation results revealed that the proposed approach is effective in obstacle avoidance tasks, recording superior time efficiency in test scenarios.

In terms of further development, we analyse the possibility to extend the testing phase by designing a methodology with a core focus on the enzymatic numerical P system models. While the current study focuses on the ENPS framework, comparing these results with other control strategies remains an important direction for future research to further validate the performance of our approach. Furthermore, we aim to validate the controllers on physical robots instances and to deepen the applicability of evolutionary algorithms in robotics, a research area that continues to be of interest, even after decades of extensive exploration.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments and suggestions and for identifying aspects of the initial version of the manuscript that required clarification, which helped improve the quality of the paper.

Author Contributions All authors contributed to this work. The first draft of the manuscript was written by Radu Traian Bobe and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Data availability All data generated or analysed during this study are included in this published article.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Asadollah, S. A., Inam, R., & Hansson, H. (2015). A survey on testing for cyber physical system. In *Testing software and systems: 27th IFIP WG 6.1 international conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23–25, 2015, Proceedings 27* (pp. 194–207). Springer.
2. Pandey, A., Pandey, S., & Parhi, D. (2017). Mobile robot navigation and obstacle avoidance techniques: A review. *International Robotics & Automation Journal, 2*(3), 00022.
3. Păun, G., & Păun, R. (2006). Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae, 73*(1–2), 213–227.
4. Pérez-Hurtado, I., Martínez-del-Amor, M. Á., Zhang, G., Neri, F., & Pérez-Jiménez, M. J. (2020). A membrane parallel rapidly-

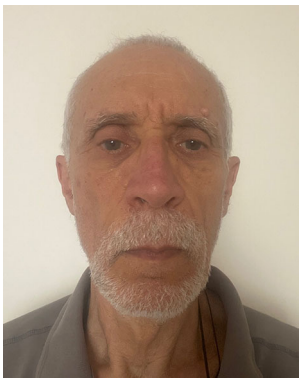
exploring random tree algorithm for robotic motion planning. *Integrated Computer-Aided Engineering, 27*(2), 121–138.
5. Bobe, R. T., Gheorghe, M., Ipate, F., & Mihai Niculescu, I. (2024). A test-driven methodology for designing robot controllers and simulators using enzymatic numerical P system models. *SIMULATION, 102*(1):45–57. https://doi.org/10.1177/00375497241292347
6. Păun, G. (2002). *Membrane Computing: An introduction* (Springer, Berlin). https://doi.org/10.1007/978-3-642-56196-2
7. Pavel, A., Arsene, O., & Buiu, C. (2010). 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. https://doi.org/10.1109/BICTA.2010.5645071
8. Liu, J., Wang, L., Zhang, G., Verlan, S., & Zhu, M. (2024). Universal enzymatic numerical P systems with small number of enzymatic rules. *Theoretical Computer Science, 1004*, Article 114630.
9. Wang, X., Zhang, G., Gou, X., Paul, P., Neri, F., Rong, H., Yang, Q., & Zhang, H. (2021). Multi-behaviors coordination controller design with enzymatic numerical P systems for robots. *Integrated Computer-Aided Engineering,* 28(2), 119–140.
10. Wang, X., Zhang, G., Neri, F., Jiang, T., Zhao, J., Gheorghe, M., Ipate, F., & Lefticaru, R. (2016). Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integrated Computer-Aided Engineering, 23*(1), 15–30.
11. Florea, A. G., & Buiu, C. (2017). PeP—an open-source software simulator of numerical P systems and numerical P systems with enzymes. [Online]. Accessed 1 May 2025. https://github.com/andrei91ro/pep
12. Harman, M., Jia, Y., & Zhang, Y. (2015). Achievements, open problems and challenges for search based software testing. In *2015 IEEE 8th international conference on software testing, verification and validation (ICST)* (pp. 1–12). IEEE.
13. Ţurlea, A., Gheorghe, M., Ipate, F., & Konur, S. (2019). Search-based testing in membrane computing. *Journal of Membrane Computing, 1*, 241–250. https://doi.org/10.1007/s41965-019-00027-w
14. Harman, M., Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR), 45*(1), 1–61.
15. Arnold, J., & Alexander, R. (2013). Testing autonomous robot control software using procedural content generation. In *Computer safety, reliability, and security: 32nd international conference, SAFECOMP 2013, Toulouse, France, September 24–27, 2013. Proceedings 32* (pp. 33–44). Springer.
16. Nguyen, C. D., Miles, S., Perini, A., Tonella, P., Harman, M., & Luck, M. (2012). Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems, 25*, 260–283.
17. Abdessalem, R. B., Nejati, S., Briand, L. C., & Stifter, T. (2018). Testing vision-based control systems using learnable evolutionary algorithms. *In Proceedings of the 40th international conference on software engineering.* ACM. *pp. 1016–1026.*
18. Gambi, A., Jahangirova, G., Riccio, V., & Zampetti, F. (2022). SBST tool competition 2022. In *Proceedings of the 15th workshop on search-based software testing.* ACM pp. 25–32. https://doi.org/10.1145/3526072.3527538
19. Humeniuk, D., Antoniol, G., & Khomh, F. (2022). AmbieGen tool at the SBST 2022 tool competition. In *Proceedings of the 15th workshop on search-based software testing.* ACM. pp. 43–46. https://doi.org/10.1145/3526072.3527531
20. Florea, A. G., & Buiu, C. (2017). Modelling multi-robot interactions using a generic controller based on numerical P systems and ROS. In *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). IEEE.* pp. 1–6. https://doi.org/10.1109/ECAI.2017.8166411
21. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., & Martinoli, A. (2009). 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.
22. Michel, O. (2004). Cyberbotics ltd. webots™: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems, 1*(1), 5. https://doi.org/10.5772/5618
23. Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). *Introduction to linear regression analysis*. Wiley.
24. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.
25. Dong, J., Zhang, G., Hu, Y., Wu, Y., & Rong, H. (2024). An optimization numerical spiking neural membrane system with adaptive multi-mutation operators for brain tumor segmentation. *International Journal of Neural Systems, 34*(08), 2450036.
26. Dong, J., Zhang, G., Luo, B., & Rong, H. (2023). An optimization numerical spiking neural P system for solving constrained optimization problems. *Information Sciences, 626,* 428–456.
27. Dong, J., Zhang, G., Wu, Y., Hu, Y., Rong, H., & Yu, T. (2024). A learning numerical spiking neural p system for classification problems. *Knowledge-Based Systems,296*, Article 111914.
28. Buiu, C., Vasile, C., & Arsene, O. (2012). Development of membrane controllers for mobile robots. *Information Sciences, 187*, 33–51.
29. Zhang, G., Pérez-Jiménez, M. J., & Gheorghe, M. (2017). *Real-life applications with membrane computing* (Vol. 25). Springer. https://doi.org/10.1007/978-3-319-55989-6
30. Zhang, G., Shang, Z., Verlan, S., Martínez-Del-Amor, M. Á., Yuan, C., Valencia-Cabrera, L., & Perez-Jimenez, M. J. (2020). An overview of hardware implementation of membrane computing models. *ACM Computing Surveys (CSUR), 53*(4), 1–38.
31. Liu, X., Zhang, G., Mastoi, M. S., Neri, F., & Pu, Y. (2023). A human-simulated fuzzy membrane approach for the joint controller of walking biped robots. *Integrated Computer-Aided Engineering, 30*(2), 105–120.
32. George, B., & Williams, L. (2004). A structured experiment of test-driven development. *Information and software Technology, 46*(5), 337–342. https://doi.org/10.1016/j.infsof.2003.09.011
33. Manikas, T. W., Ashenayi, K., & Wainwright, R. L. (2007). Genetic algorithms for autonomous robot navigation. *IEEE Instrumentation & Measurement Magazine, 10*(6), 26–31.
34. Nordin, P., & Banzhaf, W. (1995). Genetic programming controlling a miniature robot. In *Working notes for the AAAI symposium on genetic programming* (Vol. 61, p. 67). MIT, AAAI.
35. Ahuactzin, J. M., Talbi, E.-G., Bessiere, P., & Mazer, E. (1993). Using genetic algorithms for robot motion planning. In *Geometric reasoning for perception and action: Workshop Grenoble, France, September 16–17, 1991 Selected Papers* (pp. 84–93). Springer.
36. Kwaśniewski, K. K., & Gosiewski, Z. (2018). Genetic algorithm for mobile robot route planning with obstacle avoidance. *Acta Mechanica et Automatica, 12*(2), 151–159.
37. Mao, L., Ji, X., & Qin, F. (2018). A robot obstacle avoidance method based on improved genetic algorithm. In *11th International conference on intelligent computation technology and automation (ICICTA). IEEE.* pp. 327–331.
38. Binch, A., Das, G. P., Fentanes, J. P., & Hanheide, M. (2020). Context dependant iterative parameter optimisation for robust robot navigation. In *2020 IEEE international conference on robotics and automation (ICRA)* (pp. 3937–3943). IEEE.
39. Huang, W., & Lam, H. (1997). Using genetic algorithms to optimize controller parameters for HVAC systems. *Energy and Buildings, 26*(3), 277–282.

40. Saad, M. S., Jamaluddin, H., & Darus, I. Z. M. (2012). Implementation of PID controller tuning using differential evolution and genetic algorithms. *International Journal of Innovative Computing, Information and Control, 8*(11), 7761–7779.
41. Ram, A., Boone, G., Arkin, R., & Pearce, M. (1994). Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive Behavior, 2*(3), 277–305.
42. Xin, J., Zhao, H., Liu, D., & Li, M. (2017). Application of deep reinforcement learning in mobile robot path planning. In *Chinese automation Congress (CAC)* (pp. 7112–7116). IEEE.
43. Wang, B., Liu, Z., Li, Q., & Prorok, A. (2020). Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters, 5*(4), 6932–6939.
44. Anas, H., Ong, W. H., & Malik, O. A. (2021). Comparison of deep Q-learning, Q-learning and SARSA reinforced learning for robot local navigation. In *International conference on robot intelligence technology and applications* (pp. 443–454). Springer.
45. Ejaz, M. M., Tang, T. B., & Lu, C.-K. (2019). Autonomous visual navigation using deep reinforcement learning: An overview. In *2019 IEEE student conference on research and development (SCOReD)* (pp. 294–299). IEEE.
46. Vasile, C. I., Pavel, A. B., Dumitrache, I., & Păun, G. (2012). On the power of enzymatic numerical P systems. *Acta Informatica, 49*, 395–412.
47. Bobe, R. T., Gheorghe, M., Ipate, F., Niculescu, I. M. (2024). *Numerical P systems: Variants and applications.* CEUR-WS.org (Vol. 3792, p. 14). ITAT. pp. 121–128.
48. Holland, J. H. (1992). Genetic algorithms. *Scientific American, 267*(1), 66–73.
49. Bäck, T., & Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation, 1*(1), 1–23.
50. Alhijawi, B., & Awajan, A. (2024). Genetic algorithms: Theory, genetic operators, solutions, and applications. *Evolutionary Intelligence, 17*(3), 1245–1256.
51. Shukla, A., Pandey, H. M., & Mehrotra, D. (2015). Comparative review of selection techniques in genetic algorithm. In *2015 International conference on futuristic trends on computational analysis and knowledge management (ABLAZE)* (pp. 515–519). IEEE.
52. Kora, P., & Yadlapalli, P. (2017). Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications, 162*(10), 34–36.
53. Srinivas, N., & Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation, 2*(3), 221–248.
54. Fortin, F.-A., & Parizeau, M. (2013). Revisiting the NSGA-II crowding-distance computation. In *Proceedings of the 15th annual conference on genetic and evolutionary computation. ACM*. pp. 623–630
55. Yusoff, Y., Ngadiman, M. S., & Zain, A. M. (2011). Overview of NSGA-II for optimizing machining process parameters. *Procedia Engineering, 15*, 3978–3983.
56. Huang, Y., & Fei, M. (2018). Motion planning of robot manipulator based on improved NSGA-II. *International Journal of Control, Automation and Systems, 16*, 1878–1886.
57. Conesa-Muñoz, J., Ribeiro, A., Andujar, D., Fernandez-Quintanilla, C., & Dorado, J. (2012). Multi-path planning based on a NSGA-II for a fleet of robots to work on agricultural tasks. In *IEEE Congress on evolutionary computation,* (pp. 1–8). IEEE.
58. Bobe, R.T., Ipate, F., & Niculescu, I.M. (2023). Modelling and search-based testing of robot controllers using enzymatic numerical P systems. In H. Cheval, L. Leuştean, A. Sipoş (Eds.), Proceedings 7th symposium on working formal methods, Bucharest, Romania, 21–22 September 2023. arXiv e-prints, arXiv-2309.
59. Bobe, R. T., Gheorghe, M., Ipate, F., & Niculescu, I. M. (2024). Test-driven simulation of robots controlled by enzymatic numerical P systems models. In J.-L. Guisado-Lizar, A. Riscos-Núñez, M.-J. Morón-Fernández, & G. Wainer (Eds.), *Simulation tools and techniques* (pp. 56–69). Springer. https://doi.org/10.1007/978-3-031-57523-5_5
60. Floreano, D., Husbands, P., & Nolfi, S. (2008). *Evolutionary robotics*. Handbook of Robotics.
61. Savage, J., Munoz, S., Matamoros, M., & Osorio, R. (2013). Obstacle avoidance behaviors for mobile robots using genetic algorithms and recurrent neural networks. *IFAC Proceedings Volumes, 46*(24), 141–146.
62. Hamdani, T. M., Won, J.-M., Alimi, A. M., & Karray, F. (2007). Multi-objective feature selection with NSGA II. In *International conference on adaptive and natural computing algorithms* (pp. 240–247). Springer.
63. Beck, K. (2022). *Test driven development: By example*. Boston: Addison-Wesley Professional.
64. Buiu, C., & Florea, A. G. (2019). Membrane computing models and robot controller design, current results and challenges. *Journal of Membrane Computing, 1*(4), 262–269. https://doi.org/10.1007/s41965-019-00029-8
65. Humeniuk, D., Khomh, F., & Antoniol, G. (2023). Ambiegen: A search-based framework for autonomous systems testing. Science of Computer Programming, 2023, 230: 102990. https://doi.org/10.1016/j.scico.2023.102990
66. Webots reference manual. Accessed 12 Feb 2025. https://cyberbotics.com/doc/reference/proto
67. Taheri, S., & Hesamian, G. (2013). A generalization of the Wilcoxon signed-rank test and its applications. *Statistical Papers, 54*(2), 457–470.
68. Github repository. Accessed 14 Feb 2026. https://github.com/radubobe/Mobile-Robots-Optimisation/tree/670b20ec1b4714708192f251f3a56ffaf27d66bf/Git%20repository%20NSGA-II%20article
69. Meyes, R., Lu, M., De Puiseau, C. W., & Meisen, T. (2019). Ablation studies in artificial neural networks. arXiv preprint. arXiv:1901.08644; https://doi.org/10.48550/arXiv.1901.08644.
70. Hernandez, H., & Ochoa, S. (2022). Adaptive step-size one-at-a-time (OAT) optimization. *ForsChem Research Report, 7*, 1–44.
71. Luo, J., Wang, Z.-X., & Pan, K.-L. (2022). Reliable path planning algorithm based on improved artificial potential field method. *IEEE Access, 10*, 108276–108284.
72. Zhou, L., & Li, W. (2014). Adaptive artificial potential field approach for obstacle avoidance path planning. In *Seventh international symposium on computational intelligence and design* (Vol. 2, pp. 429–432). IEEE.
73. Vadakkepat, P., Tan, K. C., & Ming-Liang, W. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings of the 2000 Congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)* (Vol. 1, pp. 256–263). IEEE.
74. Rostami, S. M. H., Sangaiah, A. K., Wang, J., & Liu, X. (2019). Obstacle avoidance of mobile robots using modified artificial potential field algorithm. *EURASIP Journal on Wireless Communications and Networking, 2019*(1), 1–19.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



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



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



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