ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Second Cycle Degree in Computer Science and Engineering
Curriculum in Intelligent Embedded Systems

Mechanisms for online adaptation in robots: A comparative study

Master Thesis in:
Intelligent Robotic Systems

Supervisor
Prof. Andrea Roli

Co-Supervisor
Dr. Paolo Baldini

Candidate
Benedetta Pacilli

1^{st} Session
Academic Year 2024-2025

Abstract

Robots deployed in dynamic environments must be able to adapt autonomously to changing conditions and perturbations. This thesis examines online adaptation strategies for minimally cognitive robotic agents, with a focus on their ability to achieve and sustain high performance. We explore a range of adaptive controllers, including architectures inspired by Braitenberg vehicles and Artificial Neural Network-based strategies, from simple feed-forward topologies to recurrent networks with internal memory, each tested in navigation with a collision avoidance task. Our experimental results compare the performance of various mechanisms and adaptation policies, highlighting the trade-offs between reactivity, memory, and robustness in different online adaptation settings.

Contents

List of Figures

Introduction

Robots are increasingly required to operate in unstructured, unpredictable environments where pre-programmed behavior isn't sufficient. Classical design approaches typically rely on prior knowledge of all the possible scenarios the system might encounter. When applied to real-world contexts, characterized by noise, change, and uncertainty, this strategy falls short. To address this, research in autonomous robotics has progressively moved toward adaptive systems capable of adjusting their behavior during execution. *Online adaptation* is a promising paradigm in this area.

While in traditional *offline* learning methods, control mechanisms are optimized in simulated or limited environments and later deployed, in online adaptation, robots can modify their control policies continuously and autonomously based on real-time feedback obtained from operations. This is especially relevant in lifelong learning contexts, where the primary measure of interest is long-term cumulative performance rather than momentary optimality [BRB23]. Such mechanisms are inherently more resilient to environmental variability, internal component degradation, and unexpected interactions.

This thesis explores a range of adaptive controllers, including both neural network-based architectures and reactive designs inspired by Braitenberg vehicles, equipped with various online adaptation strategies. In particular, we investigate whether different adaptation mechanisms can achieve and maintain high performance over extended operational periods, while retaining the flexibility to respond effectively to novel or perturbed conditions.

The adaptive mechanisms presented were tested on the collision avoidance task, which was chosen for multiple reasons. First, it is one of the most used benchmarks for adaptive robots equipped with minimal cognitive capabilities. Second, it is a

continuous task without a clearly defined beginning or end, making it particularly well-suited for ongoing, online adaptation processes. So, in our case, the robot's goal is to explore an arena while avoiding collisions with walls and obstacles, receiving feedback through its proximity sensors. This minimal yet cognitively meaningful setup captures the essential challenges of sensorimotor adaptation.

Structure of the Thesis This thesis is organized into four main chapters:

- **Chapter 1 – Theoretical Background**: Illustrates the theoretical foundations of adaptive systems, focusing on online adaptation, including an overview of related works.
- **Chapter 2 – Braitenberg Architectures**: Presents and evaluates two controllers inspired by Braitenberg vehicles, comparing different sensor-to-motor mappings.
- **Chapter 3 – Neural Networks**: Discusses a range of neural network-based controllers. It starts with feed-forward architectures, progresses to recurrent ones, and explores various RNN variants, including tests with multiple robots and malfunctions.
- **Chapter 4 – Thymio Demo**: Demonstrates the application of one of the proposed adaptive strategies on a physical Thymio robot, showcasing real-world feasibility.

Chapter 1

Theoretical Background

As previously mentioned, autonomous robots are increasingly deployed in dynamic and uncertain environments. From service robotics in households to planetary exploration, we want them to be as reliable and robust as possible. In unknown or uncertain scenarios, where the conditions faced during operation may differ significantly from those assumed at design time, the ability of a robot to adapt during its lifetime becomes crucial for successful execution and long-term autonomy. Traditional robot control systems are typically designed with fixed architectures and predefined behaviors. In these cases, the robots operate effectively and meet expectations only under the specific environmental conditions for which they were intended. However, when confronted with any unexpected and unprogrammed change, such as unmodeled disturbances, hardware wear, or terrain variation, their behavior may be faulty or fail altogether. These limitations have led to a growing interest in biologically inspired, evolutionary principles. Evolutionary robotics is a subfield that draws inspiration from Darwinian evolution and seeks to automatically generate robot controllers through a process of artificial selection, mutation, and reproduction. In this context, the controllers are allowed to evolve and mutate over successive generations, guided by their fitness evaluation, rather than being hand-designed. In the work by Floreano and Keller, *Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection* [FK10], it is shown how complex adaptive behaviors can emerge in robotic systems through Darwinian selection. They demonstrate that sophisticated behaviors, such as obstacle avoid-

ance, homing, cooperation, and altruism, can arise from simple Neural Network controllers, not as a consequence of task-specific direct programming but rather as the result of selective pressures acting on random mutations in the controller's genotype, which encodes the structure and parameters of the NN. The experiments carried out in this work demonstrate the effectiveness of evolutionary methods in generating robust and adaptive behaviors capable of autonomous operation even in complex or unknown settings. For instance, their robots evolved to navigate through a looping maze without collisions, developed an intermediate-speed strategy that implicitly accounted for sensor latency, and co-evolved strategies for pursuit and evasion in a predator-prey setting, also highlighting the open-ended nature of adaptive behavior when shaped by dynamic interactions with the environment or other agents.

A first key distinction to discuss in adaptive robotic systems lies between *offline* and *online* adaptation paradigms. In offline adaptation, the robot's control policy is trained or evolved before deployment (through a wide range of techniques, such as evolutionary algorithms, supervised learning, and reinforcement learning), and the entire optimization process takes place in a simulated or controlled environment. Once a sufficiently performing controller is obtained, it is transferred to a physical robot, and from this point onward, the behavior remains fixed. Offline adaptation enables controlled experimentation and large-scale parallel optimization, but it suffers from several inherent limitations. First and foremost, performance is tightly coupled to the assumptions and fidelity of the training environment. When we move to real-world scenarios, these assumptions often break down due to intrinsic noise, unexpected dynamics, and partial observability. This mismatch between simulation and reality is known as the *reality gap*, and it isn't the only limitation. Offline methods are static and assume that the robot's hardware and environment remain constant over time. In the real world, this assumption is unrealistic, as sensors and actuators undergo aging and mechanical wear, and components may become damaged or require replacement. All these variations can significantly impair the performance of a fixed controller that lacks the ability to adapt post-deployment. The alternative to overcoming these limitations is online adaptation. Online adaptation refers to a robot's ability to autonomously modify its control strategy during operation in response to environ-

mental feedback. In this context, robot controllers are equipped with mechanisms that enable them to adjust themselves in response to changes and new experiences, both to maintain and improve their performance. Online adaptation can take various forms, each offering different levels of flexibility. A relatively simple approach is behavior switching, where the robot selects from a set of predefined controllers based on contextual cues. More flexible is parameter tuning, where internal variables (such as neural weights or decision thresholds) are adjusted online. The most general form is purely adaptive behavior, in which the robot starts with no pre-structured control and develops effective behavior solely through trial-and-error interaction with its environment. An example of purely adaptive form is provided by Baldini et al. *On the Performance of Online Adaptation of Robots Controlled by Nanowire Networks* [BRB23]. In this work, robots are controlled by nanowire networks, neuromorphic devices that are small, low-power, and structurally immutable. In fact, each nanowire network has a unique topology that cannot be precisely replicated or modified after it is manufactured. Consequently, an offline-trained controller optimized for one device cannot be directly transferred to another, forcing independent training for each robot. This renders the standard offline paradigm impractical. To address this challenge, the authors employ a purely online adaptation mechanism, where each robot begins from a state of complete ignorance, with sensor-to-network couplings initialized randomly and no predefined behaviors, learning rules, or structural priors. Adaptation occurs entirely during deployment: robots iteratively explore alternative configurations and refine them based solely on performance feedback. The authors analyzed two online adaptation strategies: *random adaptation*, which samples new configurations blindly, and *local adaptation*, which incrementally improves upon previously successful behaviors. Their results demonstrate that online strategies that maintain and refine previously effective solutions outperform those that lack memory or guidance, especially when evaluated over the robot's entire lifespan, also reinforcing the importance of treating robot adaptation as a continuous, life-long process rather than a one-time optimization. In fact, evaluating the quality of a robot's behavior based solely on momentary or peak performance can be misleading. A single high-performing episode may demonstrate that a robot is *capable* of performing a task, but it does not guarantee that it will do so *consistently*

across time and conditions. Real-world robotic systems are expected to operate autonomously for extended periods, meaning sustained performance is far more relevant than isolated success. A shift in evaluation is required, from focusing on peak performance metrics to considering cumulative or lifelong performance. Cumulative performance accounts for the total reward or score obtained over the entire operational lifetime of the robot, offering a more comprehensive view of the system's adaptability, stability, and resilience. This shift may favor strategies that are slower to reach optimality but reliably maintain acceptable performance over time, which is much more valuable in real-world deployment. Another example of this purely adaptive methodology was explored by Braccini et al. in *An Investigation of Graceful Degradation in Boolean Network Robots Subject to Online Adaptation* [BBR24], who studied how Boolean network robots can exhibit graceful degradation and structural robustness under online adaptation. In their work, the robot's control system is modeled as a Boolean network, and adaptation involves reconfiguring the couplings between sensors and the Boolean network in response to environmental changes or partial damage. The experiments demonstrate that even under adverse conditions, such as sensor failures or structural perturbations, robots can recover functionality without relying on predefined behaviors. This resilience stems solely from the inherent characteristics of online adaptation and the ability to rewire and select new configurations online.

As briefly mentioned before, the effectiveness and character of an adaptive mechanism are also influenced by the robot's embodiment and the environment in which it operates. Embodiment refers to the physical instantiation of the robot, including its sensors, actuators, morphology, and control substrate, as well as all their characteristics such as range, resolution, response time, mechanical robustness, and computational limitations. The environment, on the other hand, encompasses both the static and dynamic features of the external world that interact with the robot. Together, these two domains form the *sensorimotor loop* through which behavior is generated, evaluated, and modified. Similarly, adaptive behavior does not emerge solely from control logic; it arises from the dynamics between a robot's body and the world it acts upon. Physical constraints such as sensor delay, actuator inertia, or morphological asymmetry can shape how a robot learns, stabilizes, and refines its responses. In this view, adaptation is not simply about

adjusting internal variables but about *co-adapting* to the physical properties of the robot's body and its environment. This is well illustrated in the work of Floreano and Keller [FK10], where evolved neural controllers implicitly learn to account for limitations like sensor refresh rates and wheel momentum. For instance, in their maze navigation task, the robots had evolved to use intermediate movement speeds that indirectly compensated for sensor latency, even though no such compensation was ever explicitly programmed. This kind of fine-tuned behavioral calibration is a direct consequence of *embodiment-aware* adaptation.

One fundamental aspect of online adaptation is that at each decision point, the robot must choose between two conflicting objectives: exploiting previously successful behaviors to maintain performance, or exploring new, potentially better behaviors that have not yet been tested. A robot that only exploits may become stuck in a suboptimal behavior, while one that only explores may waste time on unproductive trials. This dilemma is known as the *exploration-exploitation trade-off*, and it is essential to manage both sides of it effectively for achieving high cumulative performance in the long run. The exploration-exploitation dilemma is formalized and explored in the *Multi-Armed Bandits* chapter from the book *Reinforcement Learning: An Introduction* by Richard Sutton and Andrew Barto [SB18]. In this setting, an agent is faced with k slot machines (*arms*), each with an unknown and potentially different reward distribution. At each time step, the agent selects an arm to pull and receives a stochastic reward. The goal is to maximize the total reward over a sequence of actions. However, since the reward distribution is initially unknown, the agent must experiment with different arms (exploration) while also remembering and reusing those known to yield higher payoffs (exploitation). Several strategies have been developed to address the Multi-Armed Bandit (MAB) problem. One of the simplest is the ϵ-*greedy* algorithm, in which the agent selects the best-known arm with probability $1 - \epsilon$ and a random arm with probability ϵ. This approach introduces controlled stochasticity, allowing exploration while still favoring high-performing actions. Another class of methods, such as the *Upper Confidence Bound* (UCB), balances exploration and exploitation by selecting arms based not only on expected reward but also on the uncertainty (or confidence interval) of their estimates. Online adaptation mechanisms in robotics can be interpreted as approximations of MAB strategies. For example, in the

work of Baldini et al. [BRB23], two contrasting approaches are proposed: *Random Adaptation* (RA) and *Local Adaptation* (LA). RA corresponds to pure exploration as it generates entirely new controller configurations at each step, without regard for past performance, similarly to a multi-armed bandit agent with $\epsilon = 1$. On the other hand, LA embodies a more balanced approach. It perturbs the best-known configuration slightly in search of nearby improvements, switching to full exploration only when current performance falls below a threshold. This mechanism is conceptually similar to an ϵ-greedy strategy with an adaptive ϵ value: the robot exploits known good configurations when possible. Still, it can fall back on full exploration when the current behavior is deemed unsatisfactory.

Many adaptation mechanisms draw direct inspiration from biology. Animals have evolved to cope with uncertainty, injury, and novel environments through a layered process of adaptation that spans evolution, development, and individual experience. Understanding how natural systems achieve this level of robustness offers valuable insights and practical design principles for building more resilient and autonomous robots. Unlike traditional engineering methods, which rely on precise models and assumptions, biological systems embrace redundancy, approximation, and trial-and-error as mechanisms of adaptation. An example of bio-inspired adaptive control in robotics can be found in the work *Robots that adapt like animals* by Cully et al. [CCM14]. In their work, a six-legged robot can recover from physical damage such as a broken leg without requiring external intervention, retraining, or reprogramming. The core of their approach is a behavior-performance map that catalogs a large number of diverse locomotion strategies, each associated with a set of behavioral descriptors and an expected performance score. During an offline phase, the map is populated with thousands of solutions that vary not only in efficiency but also in their underlying control strategies. Here, diversity is essential: it ensures that the map contains alternatives that may become useful in the face of unforeseen situations. Later, when deployed in the real world, the robot uses the map to guide a rapid trial-and-error adaptation process. Upon detecting performance degradation, the robot begins sampling candidate behaviors from the repertoire (the map), evaluating them in real-time, and converging on one that performs well given its current damaged state. This process does not require an explicit diagnosis of the damage but instead allows for behavior reorientation based

on empirical feedback. This strategy mimics the way animals attempt new movement strategies after injury, until they find one that works. The overall framework is a hybrid of *offline* and *online* adaptation. The behavior-performance map serves as a form of structured, precomputed knowledge (analogous to the innate instinctive set of behaviors in animals) while real-time feedback and behavioral evaluation drive the actual adaptation to damage. Such a design avoids many pitfalls of both extremes: pure online adaptation may require long search times or extensive trial phases that are sometimes impractical in time-sensitive tasks. On the contrary, pure offline strategies may fail entirely if their assumptions do not match the deployed conditions. Building on this theoretical background, the following chapters of the thesis present and compare multiple mechanisms for online adaptation. The primary goal is to assess their effectiveness not only in achieving peak performance but also in maintaining high cumulative performance over time and varying conditions, by analyzing a range of different architectures, exploration–exploitation strategies, and adaptation mechanisms, to identify which design principles contribute most to long-term autonomy, generalization, and fault tolerance.

Chapter 2

Braitenberg Architectures

Braitenberg Architectures [Bra86] inspired both controller mechanisms presented in this chapter.

2.1 Binary

The core of this controller mechanism is a four-bit binary configuration that directly encodes sensor-to-motor connections, allowing reactive obstacle-avoidance behavior. In the spirit of Braitenberg architectures [Bra86], the four binary weights govern the influence of two virtual proximity sensors on the two wheels. Each virtual proximity sensor is calculated by averaging three physical sensors, with three taken from the robot's left side and three from its right. While Braitenberg vehicles are typically hardcoded, in this and the following mechanism, the sensors-wheels mapping evolves over time, following online behavioral adaptation practices.

At the start of each epoch, the robot's four-bit configuration is randomly initiated, as described by the pseudocode in 1. During an epoch, at each step, the robot senses the environment and computes motor speeds using the current four-bit binary configuration. The bits define whether each sensor contributes to the left and/or right wheels, as illustrated by the schematic representation of Figure 2.1. The resulting wheel velocities determine the robot's reactive motion.

The same configuration is maintained, used, and evaluated throughout an entire epoch. At each step, the configuration is evaluated using the fitness function

Algorithm 1: Braitenberg-inspired online adaptation mechanism - Binary Version

```
Initialize config ← random 4-bit vector
best_config ← config
best_fitness ← −∞
for each epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for each step in 1 to MOVE_STEPS do
        Sense environment
        Act using current config
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum / MOVE_STEPS
    if avg_fitness > best_fitness then
        best_config ← config
        best_fitness ← avg_fitness
    end
    else
        config ← mutate(config)
    end
end
```

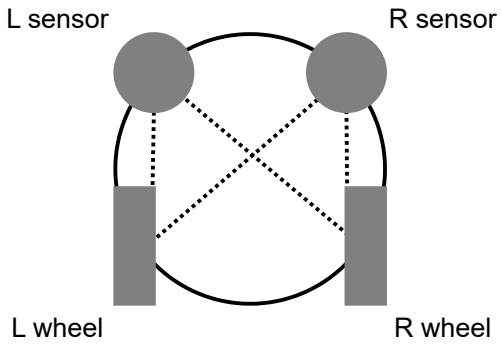

Figure 2.1: Schematic representation of the Braitenberg-inspired binary mechanism. Virtual proximity sensors on the left and right are connected to the corresponding wheel actuators via binary-weighted pathways. The four-bit configuration encodes excitatory (1) or inhibitory (0) connections between sensors and wheels.

explained in the pseudocode 2.

Algorithm 2: Fitness evaluation function

```
left_sensor_value ← get left sensor value
right_sensor_value ← get right sensor value
i ← max(left_sensor_value, right_sensor_value)
left_v_norm ← normalize_velocity(left_v)
right_v_norm ← normalize_velocity(right_v)
V ← (left_v_norm + right_v_norm)/2
return V · (1 − i)
```

This fitness function is inspired by the one proposed in the study by Floreano et al. on the evolution of adaptive behavior in robots [FK10]. The original fitness function is defined as:

$$\Phi = V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - i) \tag{2.1}$$

Where:

- V is the average normalized wheel speed;
- Δv is the absolute difference between the normalized speeds of the two wheels;
- i is the maximum activation value among the proximity sensors.

This formulation has been changed and adapted. In their work, the velocities are in the range $[-1, 1]$, where speeds in the range $[0, 1]$ correspond to rotation in one direction and speeds in the range $[-1, 0]$ correspond to rotation in the other direction [FK10]. In our case, normalized speeds are in the $[0, 1]$ range, so the $(1 - \sqrt{\Delta v})$ portion would be maximized only by using the same values of the wheels' velocities, resulting in a forward movement for the robot, regardless of obstacles. So, our version of the formula results in:

$$\Phi = V \cdot (1 - i) \tag{2.2}$$

All the fitness values calculated are cumulated at each execution step and then averaged at the end of an epoch to obtain the final average fitness produced by

a specific configuration. This average value is then evaluated against the highest (average) fitness value found so far. If the newly collected average exceeds the best one, the current average and configuration become the baseline to evaluate the next ones. Otherwise, a new candidate configuration is randomly generated by flipping a bit of the current one, as described in the pseudocode 3. This new configuration will be used and evaluated in the subsequent epoch.

Algorithm 3: Bit-flip mutation operator

```
1 i ← random index in {1, 2, 3, 4}
2 config[i] ← 1− config[i]
3 return config
```

To assess this mechanism, a total of 60 independent replicas were run. Half of these were run with uniform additive noise (± 0.01) applied to proximity sensors and wheel actuators. For each run and each epoch, we saved the epoch number, the configuration used in that epoch, and the average fitness value produced. The collected data was evaluated through a variety of metrics to understand and compare results.

All the mechanisms presented in this chapter and the subsequent one were run in the ARGoS simulator [PTO+12] using the *Footbot* robot.

Best Configuration per Run For each experimental run, the configuration that achieved the highest fitness was identified and recorded. This data produces two plots, Figure 2.2 and Figure 2.3, which enable a comparative analysis of the four-bit binary configurations that were most effective across multiple independent trials.

In both settings, specific configurations appear repeatedly among the top performers, suggesting a pattern of convergence toward effective sensor-to-motor mappings. In the noiseless condition, configuration 0110 emerges most frequently as the best across 30 runs, followed closely by 1011 and 1010. Notably, 0110 also yields the highest single-run fitness. In the noisy scenario, the overall distribution of top configurations broadens slightly. Even in this case, configuration 0110 is the most frequent, closely rivaled by 1110. Surprisingly, the configuration yielding the highest single-run fitness is 1111.

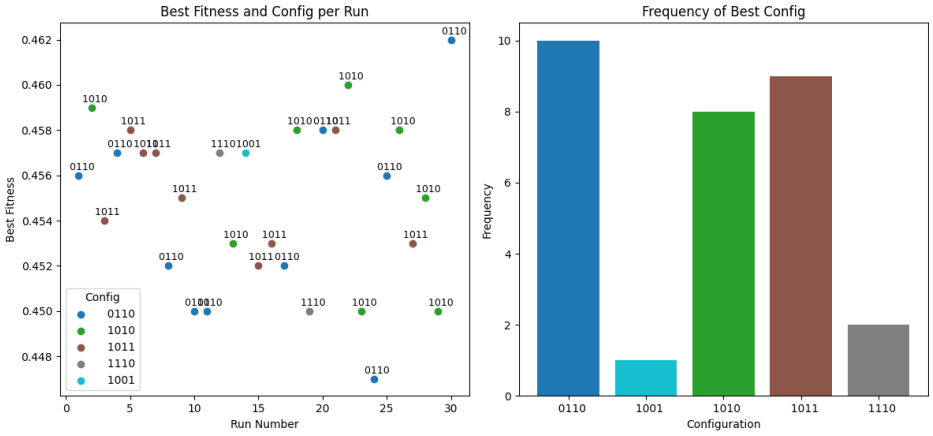

Figure 2.2: **Left**: Scatter plot showing the best fitness achieved per run along with the associated 4-bit configuration. **Right**: Bar plot displaying the frequency with which each configuration appeared as the best-performing solution across all runs.

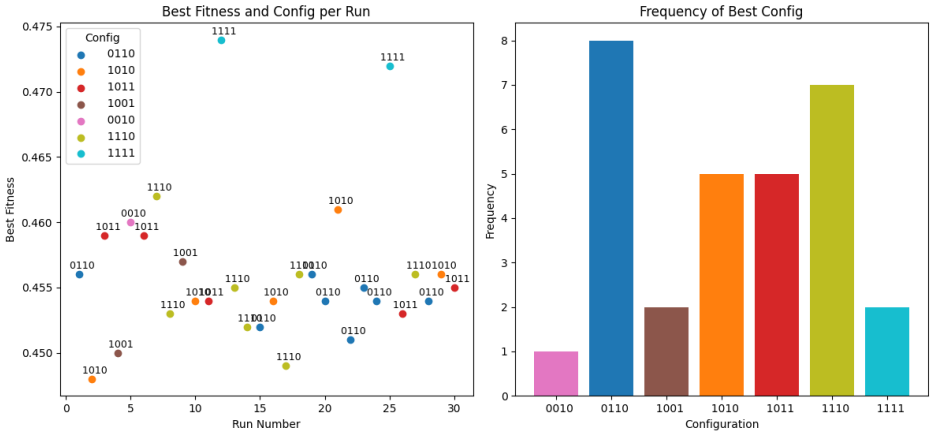

Figure 2.3: **Left**: Scatter plot showing the best fitness achieved per run (with noise) along with the associated 4-bit configuration. **Right**: Bar plot displaying the frequency with which each configuration appeared as the best-performing solution across all runs (with noise).

Since the mapping used is $\{(s_L, w_L), (s_L, w_R), (s_R, w_R), (s_R, w_L)\}$, where (s_x, w_y) means that sensor x influences wheel y, we expect the best configuration to be 1010, where each sensor influences the respective wheel only. In both scenarios, configuration 1010 is among the best, but it is neither the only one nor the absolute best one. We directly observed the best and the worst configurations in the simulation

arena, shown in Figure 2.4, to examine the quality of the resulting behaviors.

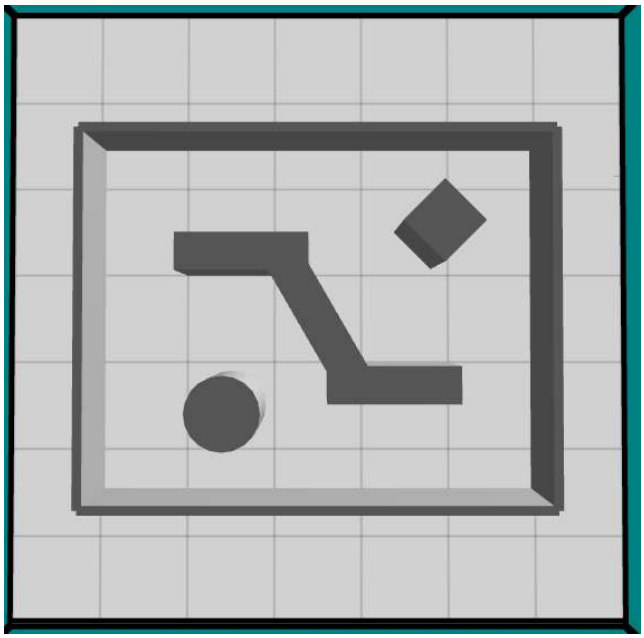

Figure 2.4: ARGoS Simulation arena. The robot moves inside the walls. The obstacles are inspired by the arena used in [FK10] for the Collision-Free Navigation experiments.

Accordingly to Figure 2.2, the most successful configurations in the noiseless setting are 0110, 1010, 1011, and 1001. Each of these exhibits a different motion strategy.

- 0110: Both left and right sensors influence only the right wheel. This causes the robot to turn left in response to any obstacle that it encounters. While effective in some situations, it lacks flexibility. In several cases, a right turn would be faster, but the robot is always forced to perform a wider, and consequently slower, turn to the left, as can be observed at second 0:54 in the simulation recording here.

- 1010: As previously said, the left sensor affects the left wheel and the right sensor affects the right wheel. This mapping enables the robot to turn away from obstacles on either side of it. However, it can struggle in frontal collision

scenarios, especially when encountering sharp angles (e.g., the 90° wall corners of the simulation arena), where it may become stuck due to symmetric influence. This behavior can be observed here.

- 1011: Both sensors influence the left wheel, and the right sensor also influences the right wheel. The robot can turn both ways, but with a stronger bias toward right turns. This results in frequent right turns and occasional imperfect left turns where the robot scrapes against obstacles and drifts along them, as shown at second 0:40 in the linked video here.

- 1001: It is the mirror configuration of 0110; both left and right sensors influence only the left wheel. This forces the robot to turn to the right in all situations, so similarly to 0110, this results in inefficient turns as the one at second 0:14 in here.

Unsurprisingly, the configurations with the lowest performance are 0000, 0001, and 0100. These all fail to produce meaningful, if any, responses to obstacle proximity.

- 0000: No sensor influences any wheel. As a result, the robot is completely unresponsive to obstacles and doesn't move, as demonstrated in this recording.

- 0001: The right sensor influences the left wheel, meaning the robot can only detect obstacles on the right and can only react by turning further to the right, colliding, or freezing entirely, as shown here.

- 0100: This produces the same asymmetric behavior as 0001, but mirrored. The left sensor affects the right wheel, causing the robot to react only to left-side stimuli by turning left. Again, this leads to collisions and unresponsive behavior.

In the noisy setting, the top-performing configurations are 0110, 1011, 1010, and 1110. Despite the introduction of noise, the first three showcase a behavior consistent with the noiseless scenario. Their resulting behaviors can be observed from the recordings 0110, 1011, and 1010 respectively. In 1110, the left sensor

influences both wheels, while the right sensor influences the right wheel exclusively. This allows for bidirectional turning but with a stronger bias towards left turns. The resulting behavior resembles the 1011 case; the robot might become stuck and drift on walls and obstacles when turning right, as it happens at second 0:25 in this recording.

The lowest-performing configurations under noise remain unchanged from the previous scenario: 0000, 0001, and 0100; their corresponding behavior is identical to the earlier setting and can be visualized in the recordings 0000, 0001, and 0100, respectively.

Fitness Statistics across Configurations The two plots, Figure 2.5 and Figure 2.6, display the average and median fitness for each of the 16 possible four-bit configurations, presenting another view on the configurations' performance.

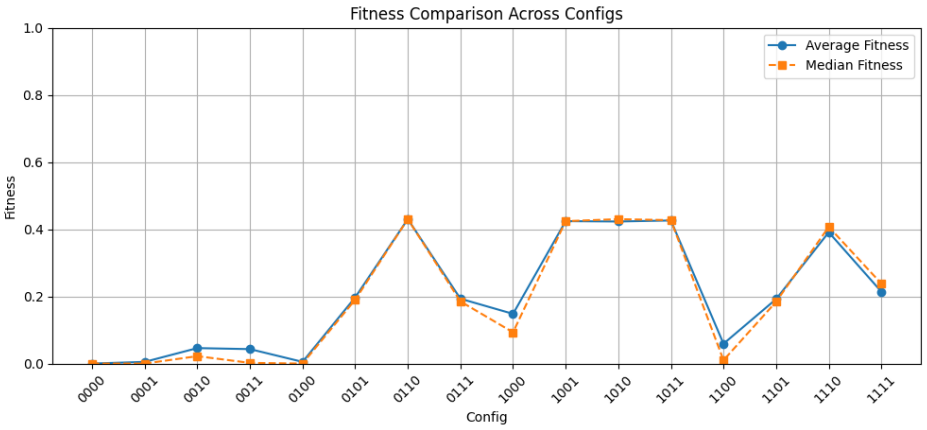

Figure 2.5: Comparison of average and median fitness across all 4-bit binary configurations.

In both contexts, 0110, 1001, 1011, 1010, and 1110 consistently achieve the highest values. These configurations enable balanced and reactive behaviors, such as the ability to turn in both directions or respond effectively to obstacle proximity on either side. On the contrary, 0000, 0001, 0100, and 1100 yield the poorest statistics, as expected.

Interestingly, the statistical results stay consistent between the noiseless and noisy conditions. However, in the noisy case (Figure 2.6), several configurations

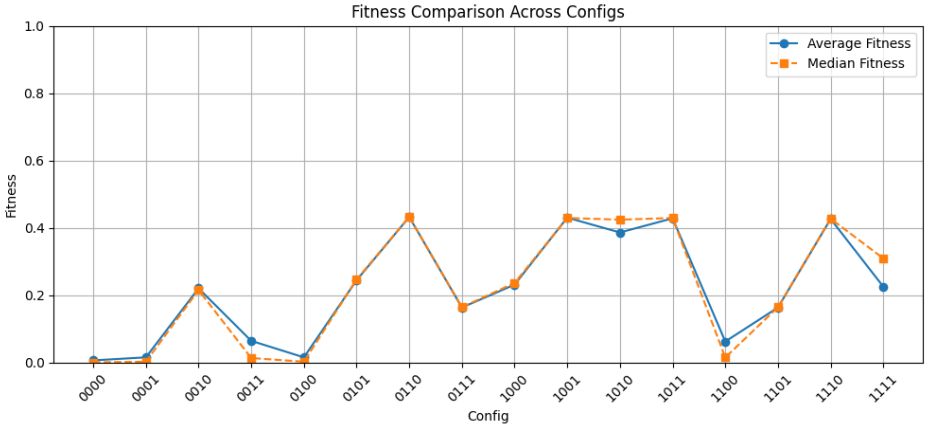

Figure 2.6: Comparison of average and median fitness across all 4-bit binary configurations, from noisy data.

like 0010, 1000, and 1111 display slightly elevated medians compared to the noiseless scenario, suggesting that some configurations may be more robust or flexible in the presence of noise.

Average Fitness over Time Figure 2.7 shows two curves, representing the evolution of the average fitness across all epochs, under noiseless and noisy conditions.

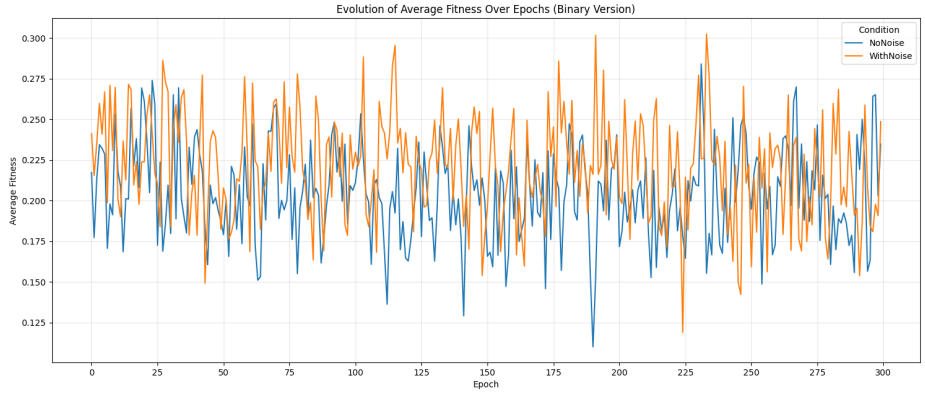

Figure 2.7: Evolution of average fitness across epochs for binary Braitenberg mechanism under noise and no-noise conditions.

Although the gap between the two curves is small, the first thing noticeable is that the orange line, representing noisy setting data, exhibits more frequent

and higher peaks. This may suggest that noise avoids stagnation and promotes flexibility.

A notable observation in both cases is the high variance in the curves. This is inherent to the nature of the binary controller, where a single bit flip in the sensor-motor mappings can produce significantly different behaviors.

Mean Cumulative fitness Figure 2.8 presents the evolution of the mean cumulative fitness over 300 adaptation epochs. The cumulative metric captures the total reward (fitness) accumulated over time, thus reflecting the lifelong performance of each approach.

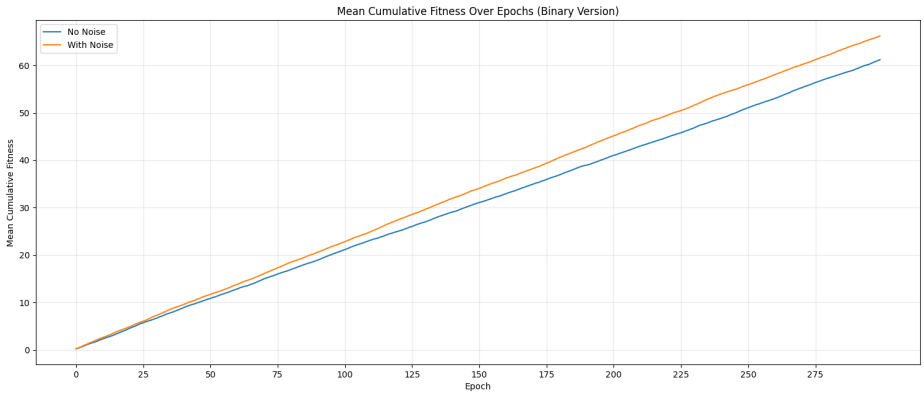

Figure 2.8: Mean cumulative fitness across epochs for binary Braitenberg mechanism under noise and no-noise conditions.

The curves indicate a performance advantage for the noisy condition, with the corresponding curve consistently above that of the noiseless counterpart. Despite the fluctuation previously seen in Figure 2.7, both curves suggest successful behaviors in both cases, with the noisy setting again being at an advantage.

Final Fitness Sum Distribution Figure 2.9 offers a comparison of the total fitness accumulated over all epochs for each replica in the noiseless and noisy conditions. This highlights not only the median performance but also the variability and range of outcomes across independent runs.

The box plots have a similar inter-quartile range[1] (IQR) width; however, the

[1]The IQR represents the middle 50% of the data, spanning from the first quartile (Q1, below

noisy box plot exhibits a noticeably higher median total fitness. This metric reinforces what has been previously shown: noise, rather than impairing adaptation, appears to enhance it.

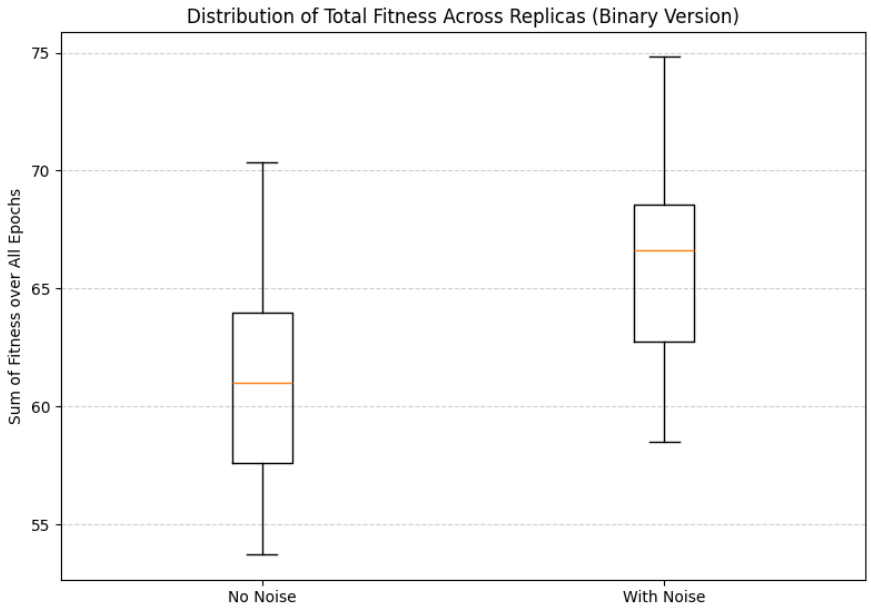

Figure 2.9: Distribution of total fitness across replicas under noise and no-noise conditions (binary Braitenberg mechanism).

Maximum Fitness Distribution Unlike cumulative fitness, which captures overall performance across all epochs, this metric focuses on each replica's best-performing moment, presenting the distribution of the highest fitness value achieved by each replica during the entire run.

Figure 2.10 shows two similar plots in terms of median, while the box plot spread of the noiseless scenario is wider both in the upper and lower ends.

Cumulative % of Successful Replicas Finally, Figure 2.11 illustrates the cumulative percentage of replicas that reached or exceeded a fitness threshold of 0.40.

which 25% of the data fall) to the third quartile (Q3, below which 75% of the data fall). It indicates the spread of the central values and is visualized by the height of the box. A smaller IQR suggests more consistency across runs, while a larger IQR indicates greater variability.

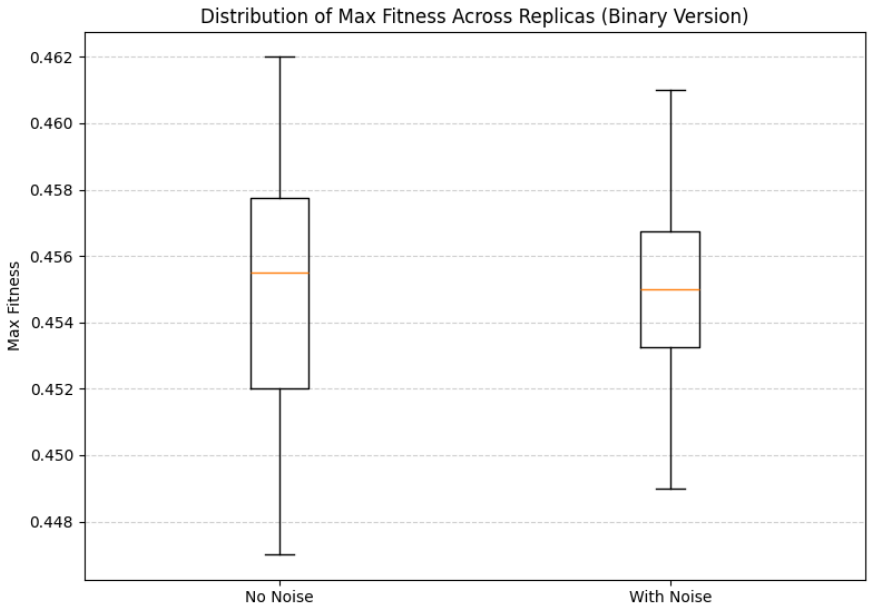

Figure 2.10: Distribution of maximum fitness across replicas under noise and no-noise conditions (binary Braitenberg mechanism).

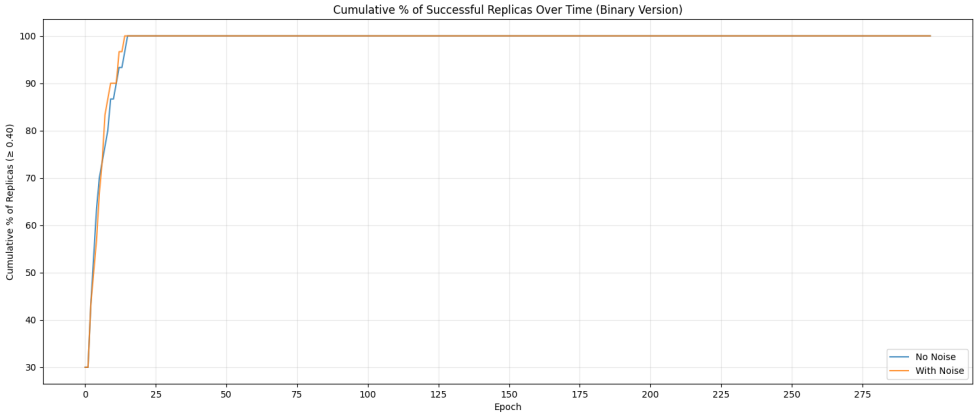

Figure 2.11: Cumulative percentage of successful replicas over time under noise and no-noise conditions (binary Braitenberg mechanism).

The threshold was chosen by directly observing the robot in the simulation arena and concluding that a configuration yielding a fitness value above 0.40 results in a satisfactory obstacle-avoidance behavior.

Both scenarios eventually reach 100% success, indicating that all replicas, regardless of noise, are capable of discovering effective configurations if given suffi-

cient time. If we consider a narrow window going approximately from epoch 5 to epoch 10, we can observe that the noiseless curve lies above the other one. Past this window, the noisy curve dominates for the remaining epochs.

2.2 Continuous

The second and last mechanism inspired by Braitenberg Architectures [Bra86] shares many design principles with the first. While in the binary case (Section 2.1), the sensor-to-motor mapping is a four-bit binary configuration, in the continuous case, the mapping is represented by four real-valued parameters in the range $[0, 1]$, where 0 is the total inhibition of a particular connection and 1 represents the maximum activation of a connection. Real-valued parameters allow a more fine-grained modulation of the connections, offering a broader and richer space of possible configurations.

Apart from the initialization and mutation of configurations, the algorithm, observable from Pseudocode 4, stays identical to the binary one, including the fitness evaluation.

Configurations are muted by introducing a small perturbation $\delta \in [-0.2, +0.2]$ to one of the four parameters, chosen randomly, as formally described in Algorithm 5. Unlike binary mutation, this operator performs a bounded additive adjustment to simulate continuous variation. After the selected parameter is mutated, it is clamped back into the $[0, 1]$ range to ensure the fitness calculation remains valid.

This mechanism was evaluated similarly to the binary case; 60 independent runs were executed using the same arena (see Figure 2.4). Half of these were run with uniform additive noise (± 0.01) applied to proximity sensors and wheel actuators. The same data was saved and later analyzed using a similar set of metrics.

Best Configuration per Run For each independent experiment, we identify and record the best configuration in terms of fitness value. As previously mentioned, for this mechanism, configurations are composed of real-valued numbers, thereby increasing our total number of possible configurations from 16 to infinity. Many of these configurations can be considered similar in terms of their resulting

Algorithm 4: Online adaptation for Braitenberg continuous mechanism

```
config ← 4 random real-valued parameters ∈ [0, 1]
best_config ← config
best_fitness ← −∞
for epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for step in 1 to MOVE_STEPS do
        sense environment
        act using config
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum / MOVE_STEPS
    if avg_fitness > best_fitness then
        best_config ← config
        best_fitness ← avg_fitness
    end
    else
        config ← mutate(config)
    end
end
```

Algorithm 5: Continuous configuration mutation

```
i ← random index in {1, 2, 3, 4}
δ ← random float in [−0.2, +0.2]
config[i] ← clamp(config[i] + δ, 0.0, 1.0)
return config
```

behavior; e.g., $\{0, 0, 0.99, 0.60\}$ is very similar to $\{0, 0.05, 0.89, 0.59\}$. So to gain comprehensive insights on which configurations, or better, which *kinds* of configurations, performed better, we first approximate each configuration number to its closest integer (0 or 1). Formally, each number in the $[1, 0.5]$ range is approximated to 0 while numbers in the $(0.5, 1]$ range to 1. Results from this analysis are shown in Figure 2.12 and Figure 2.13.

Different configuration groups appeared from the noiseless scenario, listed in Figure 2.12. Among the most frequent best configurations, we find some that also appear in the binary case, such as 0110, 1010, and 1011. Looking at the scatter plot

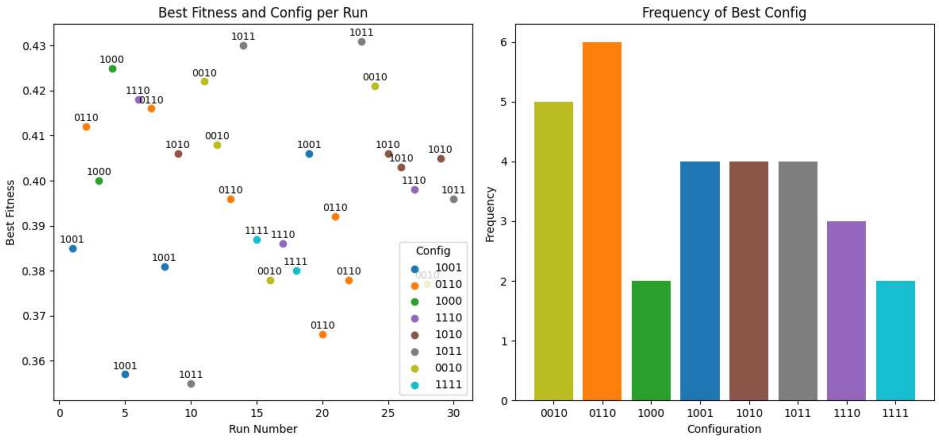

Figure 2.12: **Left**: Scatter plot showing the best fitness achieved per run along with the associated (approximated) 4-bit configuration. **Right**: Bar plot displaying the frequency with which each (approximated) configuration appeared as the best-performing solution across all runs.

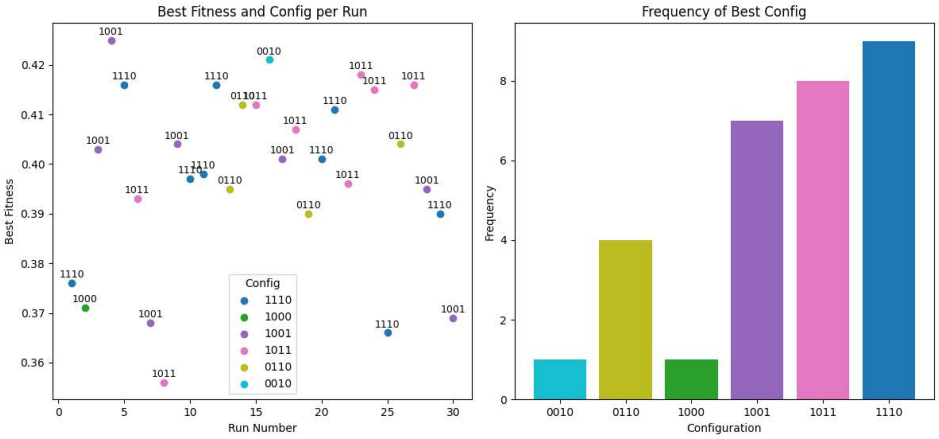

Figure 2.13: **Left**: Scatter plot showing the best fitness achieved per run (with noise) along with the associated (approximated) 4-bit configuration. **Right**: Bar plot displaying the frequency with which each (approximated) configuration appeared as the best-performing solution across all runs (with noise).

on the left, good fitness results were achieved by all, especially 1010, which stayed the most consistent. Interestingly, one of the best configuration groups found is 0010, meaning only the right sensor influences the right wheel. 0010 delivers high fitness values and is the second most frequent configuration group.

In the noisy scenario (Figure 2.13), the most frequent best configuration groups are 1110, 1011, and 1001, which all returned good fitness values, in a range similar to the noiseless case. The configuration group 0010 appeared again as one of the best, but at a significantly lower frequency.

The behaviors produced by the best and worst configurations found for this mechanism were directly observed as well, in the same arena (Figure 2.4) and simulation conditions as the previous mechanism. This time, not the approximations but the real configurations were put under analysis.

From the noiseless setting, the best configurations found are 0.99 0.15 0.80 0.74, 0.79 0.65 0.95 0.17, 0.85 0.66 1.00 0.09, 0.80 0.58 1.00 0.00.

- 0.99 0.15 0.80 0.74: This configuration produces a stronger influence on the left wheel, due to the high values on the first and fourth weights (0.99 and 0.74). The robot can turn in both directions, although its turning bias leans to the right. The resulting behavior closely resembles that of 1011: right turns are dominant, with occasional imprecise left turns, during which the robot collides and drifts along obstacles, as it happens at second 0:23 in here.

- 0.79 0.65 0.95 0.17: Functionally similar to the previous case, this setup exhibits a stronger influence on the right wheel, while still allowing for bidirectional turning, similarly to 1110. Turns to the left are prevalent, alongside less frequent and less precise right turns. This behavior is captured by this recording

- 0.85 0.66 1.00 0.09 and 0.80 0.58 1.00 0.00: Among all best configurations, these two are the most similar (in behavior) to 1010. The second of the two (0.80 0.58 1.00 0.00) appears to be the most stable and effective, and unlike 1010, it does not get stuck in sharp corners, as showcased at second 0:55 in here. In contrast, 0.85 0.66 1.00 0.09 may present the hitting and scraping obstacles behavior observed in less balanced configurations. This last behavior is captured by this recording.

There are numerous configurations registered as worst, but many of them can be considered similar to each other, in form and delivered behavior. As an example, the following were chosen: 0.00 0.00 0.03 0.85, 0.48 0.80 0.00 0.00, and

1.00 0.65 0.99 0.46. In the first two cases, the robot is unresponsive to obstacles, freezing completely. In the third and final configuration example, due to the medium and high influence on all wheels from all sensors, the robot often bumps into obstacles, as in this simulation recording.

In the noisy scenario, the following configurations emerged as best:

- 0.48 0.83 0.91 0.00: The highest values (0.83 and 0.91) represent the influence on the right wheel. With this configuration, the robot can only turn left. It might attempt a right turn but without succeeding; the robot begins the right turn, then bumps into the obstacle and finishes with a left turn. This can be viewed at second 0:08 in here.

- 1.00 0.12 0.37 0.80: This is the opposite scenario of the previous configuration. The left wheel has the most influence, and the robot can only turn to the right. Since the right wheel is slightly influenced by the right sensor, when sensing an obstacle on the right side, it may try to turn to the left, without success. When turning left is more convenient, the robot attempts to do so, but it still ends up turning right, often after bumping into the obstacle. The resulting behavior of this configuration is recorded here.

- 1.00 0.00 0.93 0.31: This case is very similar to 1010. The robot can turn both left and right (as shown here) and, unlike 1010, it does not get stuck in corners.

- 0.91 0.13 0.89 0.76: Finally, with this configuration, it can turn both ways, but the influence is stronger on the left wheel, so when trying to turn left, it bumps and drifts away as in 1011, as it happens at second 0:01 in this video.

The configurations labeled as worst from the experiments with noise are similar to those found in the noiseless scenario: the majority of the worst configurations have almost all elements close to 0.00; in these cases, the robot does not move at all when detecting an obstacle. A case different from these but still equally bad is 0.71 0.82 0.70 0.84; both wheels have a similarly strong influence. This causes the robot to freeze, unable to decide which way to turn when encountering obstacles, as it happens here.

Correlation between Configuration Fitness and Distance from Optimal (1010) Another observation that can be made on the continuous configurations obtained and the one considered as the optimal best is given by the four scatter plots in Figure 2.14 and Figure 2.15.

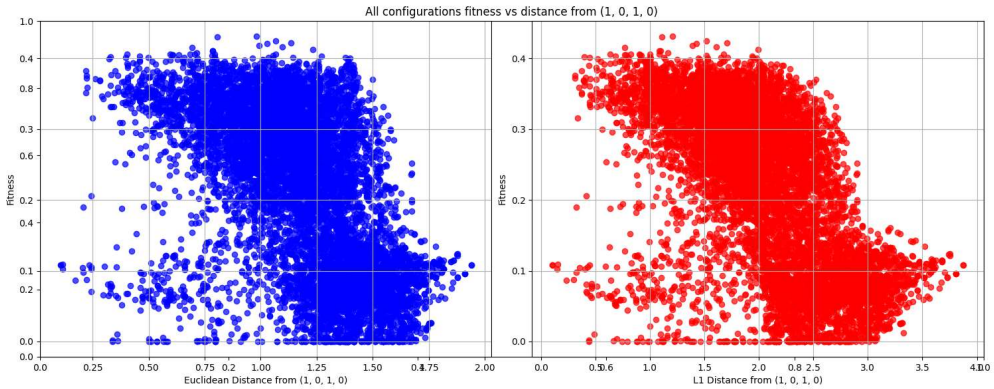

Figure 2.14: Fitness values plotted against Euclidean (left) and Manhattan (right) distances from the optimal binary configuration $(1, 0, 1, 0)$.

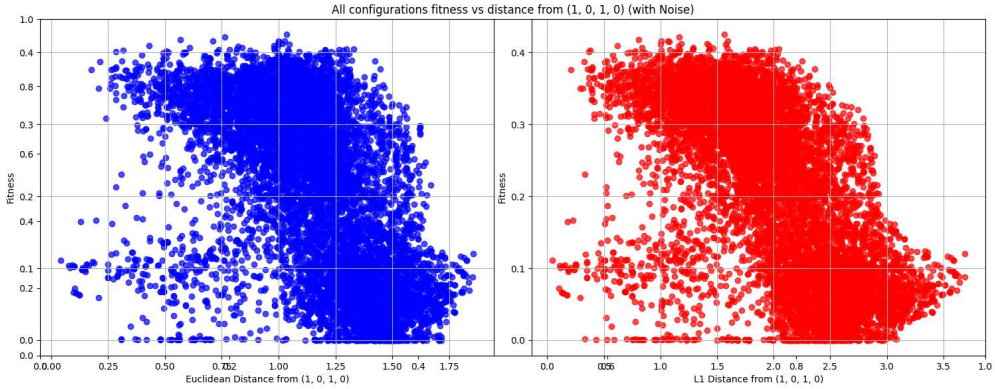

Figure 2.15: Fitness values (collected in noisy scenario) plotted against Euclidean (left) and Manhattan (right) distances from the optimal binary configuration $(1, 0, 1, 0)$.

These scatter plots show the fitness achieved by continuous configuration controllers as a function of their distance from the optimal binary configuration 1010, in both noisy and noiseless settings. In each case, the left panel uses Euclidean

(L1) distance, while the right panel uses Manhattan (L2) distance [2]. Each point represents a single configuration trial during an epoch.

The plots examine whether configurations that are numerically closer to the optimal binary configuration also tend to yield higher fitness values. Both distances show a weak, though visible, negative correlation: in general, configurations farther from the reference point tend to perform worse. This suggests that the behavior encoded by 1010 lies near a local maximum in the configuration space. A few outlier configurations at moderate distances still achieve high fitness, indicating that while proximity to 1010 is a useful heuristic, it is not a strict requirement for good performance. These outliers may represent alternative local optima.

Average Fitness over Time Figure 2.16 showcases the evolution of average fitness across epochs, in both noisy and noiseless experimental settings.

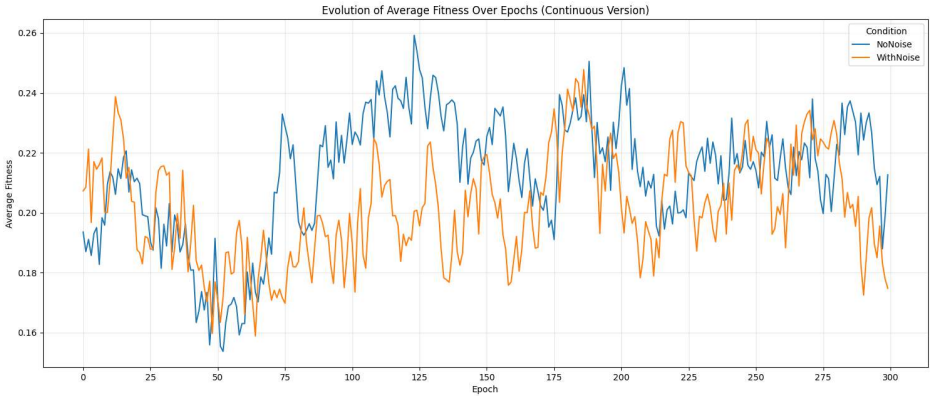

Figure 2.16: Evolution of average fitness across epochs for continuous mechanism under noise and no-noise conditions.

The learning curves reveal high variance in both conditions, but with slightly different trends. In the early stages, the noisy case shows a slight advantage, with curve peaks higher than the other. In the mid-epochs, the noiseless scenario curve constantly dominates, whereas in the later stages, the two curves alternate in dominance. Notably, the noisy curve starts higher but finishes at a significantly

[2]Manhattan (L1) distance sums the absolute differences across dimensions, while Euclidean (L2) distance computes the square root of the sum of squared differences, giving more weight to larger deviations.

lower fitness value, while the noiseless one reaches a final higher value, despite starting at a lower level.

If we compare this plot (Figure 2.16) with the corresponding binary one (Figure 2.7), we can state that in both cases, high variance is evident. In the continuous mechanism, fitness improves more steadily, especially under the noiseless condition, which ultimately outperforms the noisy one. In contrast, the binary version shows no clear long-term winner, even if the noisy condition exhibits more frequent and higher peaks.

Mean Cumulative Fitness The plot illustrated in Figure 2.17 tracks the accumulation of fitness over time, describing long-term performance in both noisy and noiseless scenarios.

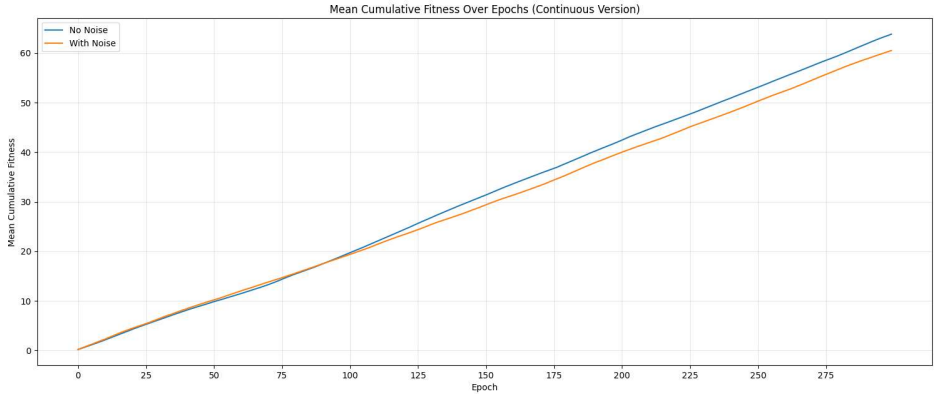

Figure 2.17: Mean cumulative fitness across epochs for continuous Braitenberg mechanism under noise and no-noise conditions.

Both conditions show steady improvement, but the noiseless setting eventually accumulates more fitness than the noisy one, especially in the later stages. During the first ≈ 90 epochs, the orange curve, representing the noisy scenario, prevails while later the situation inverts and the gap gradually widens. This reinforces the previously mentioned insight that, for the continuous mechanism, noise is beneficial initially but detrimental in the long run when stable optimization is required.

Compared to the binary mechanism (Figure 2.8), where the noisy curve is consistently above, final results are inverted. In the binary case, the noiseless curve reaches just above the mean cumulative fitness value of 60 while the other

curve (noisy) reaches around 65. On the contrary, in the continuous case, the noisy curve is just above 60 while the other curve (noiseless) reaches around 65.

Final Fitness Sum Distribution In Figure 2.18, the box plots offer a different view of the cumulative fitness score.

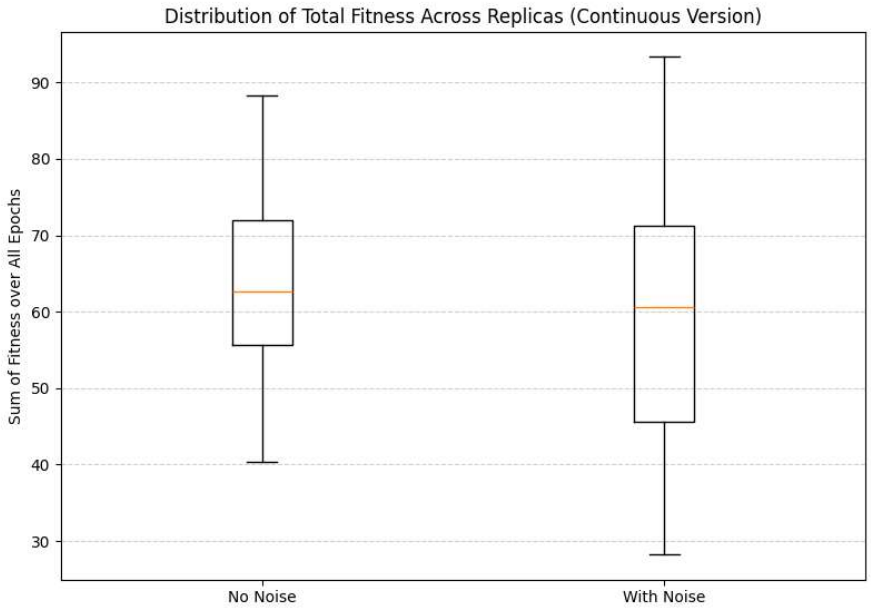

Figure 2.18: Distribution of total fitness across replicas under noise and no-noise conditions (continuous Braitenberg mechanism).

The noiseless box plot presents a slightly higher median and reduced spread and IQR, compared to the noisy one. Wider ranges and more extreme values, both on the lower and higher end, suggest that noise here introduces more variability across runs, increasing the likelihood of both strong and weak outcomes.

Maximum Fitness Distribution The plot in Figure 2.19 compares the best fitness values reached by each mechanism across all runs.

The two conditions exhibit very similar distributions, with nearly identical medians, similar spreads, and overlapping inter-quartile ranges, meaning that both conditions are equally capable of discovering high-performing solutions. A similar result is also achieved in the binary case, as can be observed from Figure 2.10.

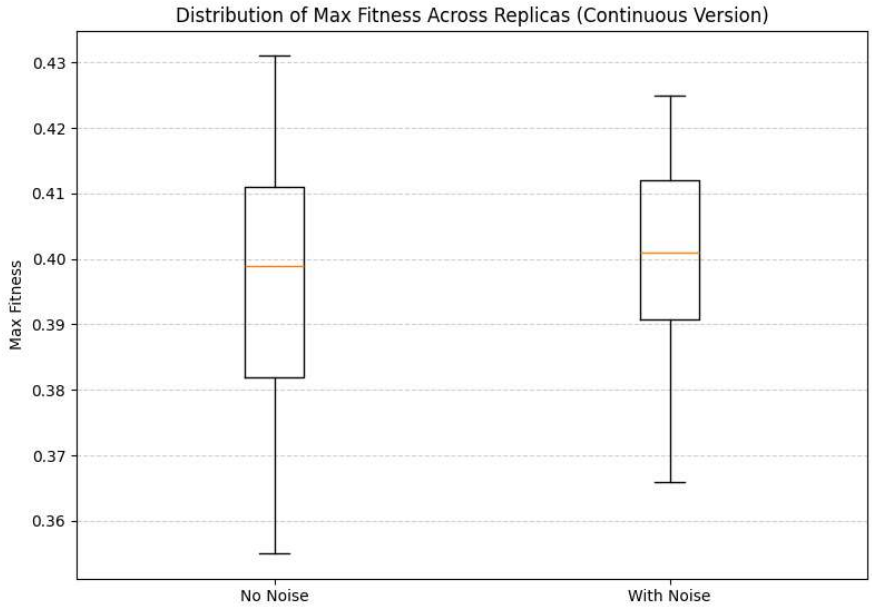

Figure 2.19: Distribution of maximum fitness across replicas under noise and no-noise conditions (continuous Braitenberg mechanism).

Cumualtive % of Successful Replicas Figure 2.17 illustrates the cumulative percentage of replicas that reached a fitness of at least 0.40 at any point during their execution.

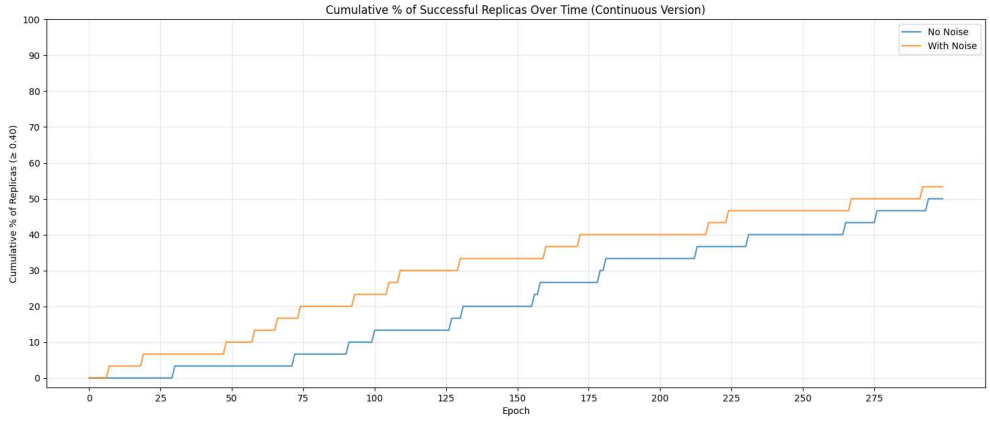

Figure 2.20: Cumulative percentage of successful replicas over time under noise and no-noise conditions (continuous Braitenberg mechanism).

By observing the curves, we can see that neither the noisy nor the noiseless

condition reaches 100% of successful replicas, but both curves reach at most $\approx 50\%$ during their last epochs. We can also observe that throughout all the epochs, the noisy curve consistently dominates.

This result contrasts with the one obtained with the binary controllers (Figure 2.8). In the binary version, all replicas reach full convergence extremely quickly, within the first ≈ 20 epochs, regardless of noise presence. In contrast, the continuous version exhibits a much slower and more gradual increase, yet never achieves complete success.

Chapter 3

NNs

The mechanisms presented and evaluated in this section differ from the previous binary and continuous strategies. Previously, obstacle avoidance behavior emerged through a direct mapping between sensor inputs and motor outputs. The controllers described here, on the other hand, use artificial neural networks (ANNs), introducing an additional level of processing between sensing and actuation.

3.1 Feed-Forward Neural Network

The mechanism presented in this section is the first to be implemented through a Neural Network. The core idea remains the same: the robot senses its environment and, over time, must find the optimal configuration to avoid obstacles. While in the previous two mechanisms, the task of obstacle avoidance was achieved through direct mappings from sensors to wheel velocities, in this network-based mechanism, the internal processing differs. Instead of a fixed or parametrically tunable sensor-to-motor mapping encoded in a compact vector, here the robot's decisions emerge from a layered neural architecture transforming sensory inputs through weighted connections. These weights are the parameters of the neural network, which are learned and optimized over time.

The basis of the whole mechanism can be observed in Pseudocode 6. The controller is a standard Feed Forward Neural Network with optional hidden layers. The input layer receives two inputs: the virtual proximity sensor values for the

left and right sides. The output layer produces the two values used to control the robot's left and right wheel velocities. If a hidden layer is present, it contains H neurons. The network parameters are stored in a real-valued vector θ, of size `PARAMS`, initialized with random values in the range $[-1, 1]$. The number of parameters depends on the network topology:

- **No hidden layer ($H = 0$):** the network directly connects the two inputs to the two outputs, with each output having its pair of input weights and one bias (as can be observed in Figure 3.1), resulting in $2 \times 2 + 2 = 6$ total parameters.

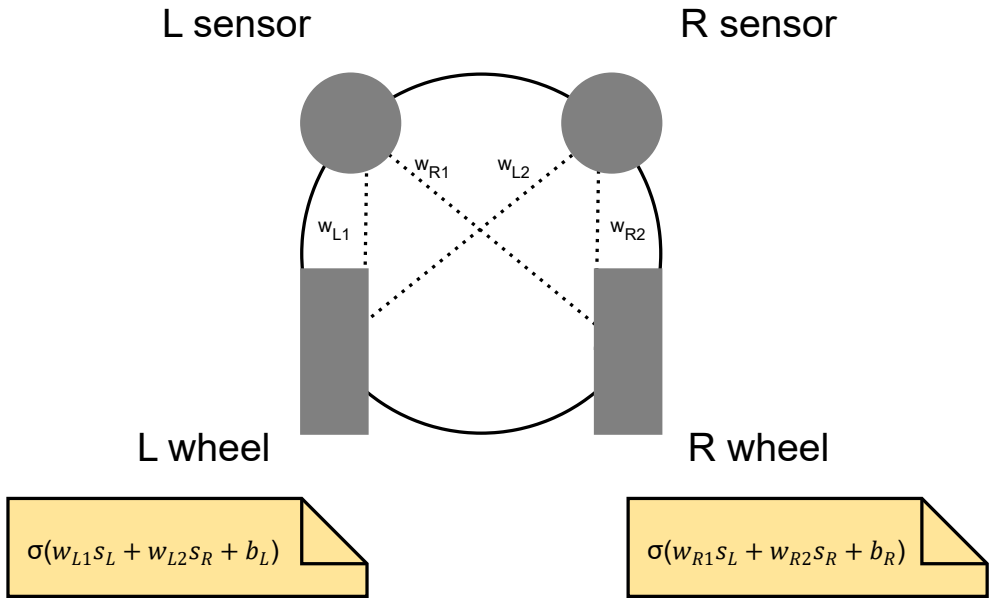

Figure 3.1: Schematic representation of the feed-forward NN controller with no hidden layer ($H = 0$). Each virtual proximity sensor influences both wheel actuators through real-valued weighted connections. The wheel speeds are computed as sigmoid-activated weighted sums of the two sensor inputs, plus a bias term.

- **With hidden layer:** the parameter count increases to account for input-to-hidden weights, hidden biases, hidden-to-output weights, and output biases:

$$\texttt{PARAM\_COUNT} = I \cdot H + H + H \cdot O + O$$

where $I = 2$ (inputs), H is the number of hidden neurons, and $O = 2$ (outputs). The schematic representation of this case can be visualized in Figure 3.2.

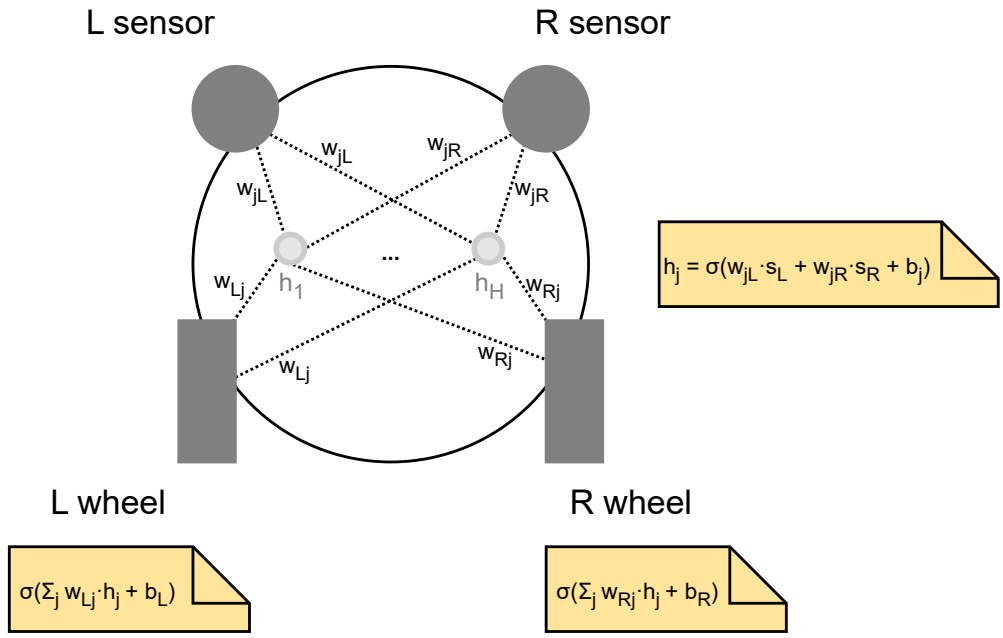

Figure 3.2: Schematic representation of the general feed-forward NN controller with a hidden layer ($H > 0$). Each hidden neuron receives input from both sensors and computes an activation value through a weighted sum and a bias, followed by a sigmoid. The wheel velocities are obtained by aggregating the activations of all hidden neurons using another set of weights and biases, followed by a final sigmoid activation.

The robot executes a complete epoch by using a fixed configuration (θ), which encodes all neural network weights and biases. At each epoch, the robot senses the environment, and the obtained sensor readings are fed into the NN (see Pseudocode 6), which outputs motor velocities based on the current θ. The resulting motion translates into a behavior maintained throughout an entire epoch and evaluated at each step, using the same fitness function described in Algorithm 2. The fitness values obtained at each step are cumulated during an epoch and averaged at the end of it. If the current θ outperforms the configuration previously found as best, it is retained as the new best and will be used as baseline in the successive epochs,

until overthrown. Otherwise, the current configuration is mutated by selecting a random parameter in θ and modifying it by adding a small random δ, as described in pseudocode 7. The newly obtained configuration will be used and consequently evaluated in the following epoch.

Algorithm 6: Online adaptation for Feed Forward NN mechanism

```
θ ← vector of PARAMS random real-valued weights ∈ [−1, 1]
best_θ ← θ
best_fitness ← −∞
for epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for step in 1 to MOVE_STEPS do
        sense environment → (s_L, s_R)
        output ← FFNN_forward(s_L, s_R, θ)
        set motor speeds using output
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum / MOVE_STEPS
    if avg_fitness > best_fitness then
        best_θ ← θ
        best_fitness ← avg_fitness
    end
    else
        θ ← mutate(θ)
    end
end
```

Algorithm 7: `mutate` function for perturbing one weight

```
Function mutate(θ):
    i ← random index in [1, PARAMS]
    δ ← random float in [−0.5, +0.5]
    θ[i] ← clamp(θ[i] + δ, −1.0, 1.0)
    return θ
```

The neural forward pass, described in Algorithm 8, operates differently depending on the presence of hidden neurons:

- For $H = 0$, output neurons are computed as:

$$z_L = w_{L1} \cdot s_L + w_{L2} \cdot s_R + b_L, \quad \text{then out}_L = \sigma(z_L)$$

  With an equivalent formulation for the right wheel.

- For $H > 0$, each hidden neuron computes an activation value:

$$h_j = \sigma(w_{jL} \cdot s_L + w_{jR} \cdot s_R + b_j)$$

  And the output neurons then compute the output values used to calculate wheel velocities:

$$\text{out}_L = \sigma\left(\sum_j w_{Lj} \cdot h_j + b_L\right)$$

  with a symmetric equation for out_R.

Algorithm 8: Feed Forward function for computing motor outputs

Function `FFNN_forward`(s_L, s_R)**:**
  if $H = 0$ **then**
    $z_L \leftarrow w_{L1} \cdot s_L + w_{L2} \cdot s_R + b_L$
    $z_R \leftarrow w_{R1} \cdot s_L + w_{R2} \cdot s_R + b_R$
    $out_L \leftarrow$ `sigmoid`(z_L)
    $out_R \leftarrow$ `sigmoid`(z_R)
  end
  else
    for j *in* 1 *to* H **do**
      $z \leftarrow w_{jL} \cdot s_L + w_{jR} \cdot s_R + b_j$
      $h_j \leftarrow$ `sigmoid`(z)
    end
    $z_L \leftarrow \sum_j w_{Lj} \cdot h_j + b_L$
    $z_R \leftarrow \sum_j w_{Rj} \cdot h_j + b_R$
    $out_L \leftarrow$ `sigmoid`(z_L)
    $out_R \leftarrow$ `sigmoid`(z_R)
  end
  return (out_L, out_R)

This NN-based mechanism was tested and assessed, in the same arena as before

(Figure 2.4), using different hidden layer (H) sizes. For each hidden layer size tested, a total of 60 independent runs were executed; half of these runs added uniform additive noise (± 0.01) to the proximity sensors and wheel actuators. For each epoch of each run, we saved the epoch number with the corresponding fitness value produced. The acquired data was studied using a variety of suitable metrics.

Average Fitness Over Time Figure 3.3 and Figure 3.4 present the evolution of average fitness across epochs for varying numbers of hidden units H, under both noiseless and noisy conditions.

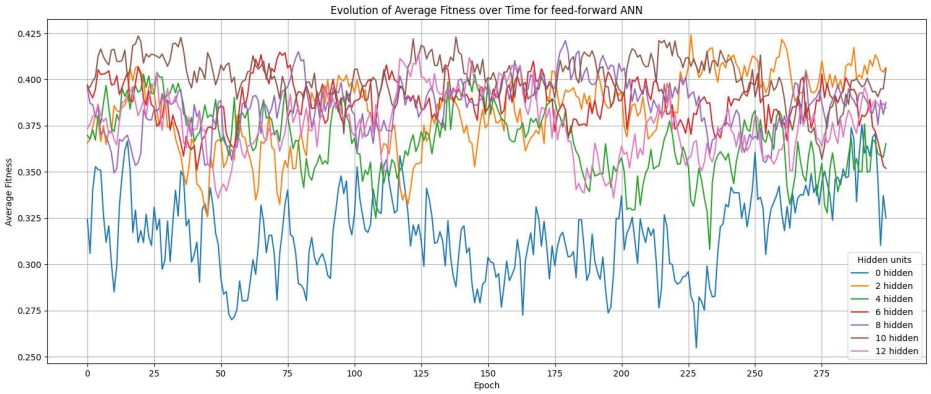

Figure 3.3: Evolution of average fitness across epochs for Feed Forward NN-based mechanism under no-noise condition, with different hidden layer sizes.

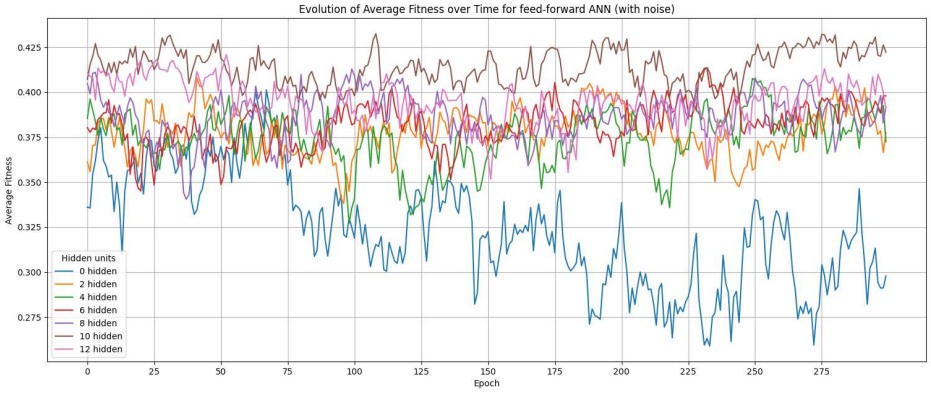

Figure 3.4: Evolution of average fitness across epochs for Feed Forward NN-based mechanism under noise condition, with different hidden layer sizes.

In both cases, the smallest controller with $H = 0$ has the worst performance. Its average fitness starts lower than the others and either stagnates or degrades even further in the presence of noise. In contrast, the best-performing H size is consistently $H = 10$, which quickly rises above 0.40 and maintains stable high values across most epochs, even in the presence of noise.

Intermediate values of H (e.g., $H = 2$ and $H = 4$) reach moderate to high peaks during learning but show less consistency, frequently oscillating and failing to retain optimal behavior throughout the run. The curves for $H = 6$ and $H = 8$ show more stable trends compared to lower H, especially in the noiseless case, but they still fail to match the peak and sustained performance of $H = 10$.

$H = 12$ performs well in the early stages of the noisy scenario, but it never reaches the $H = 10$ performance. While in the noiseless setting, it goes through some high peaks, but it appears very unstable. This can be due to overfitting and increased network complexity.

Mean Cumulative Fitness Mean cumulative fitness trend is captured by the plots of Figure 3.5 and Figure 3.6.

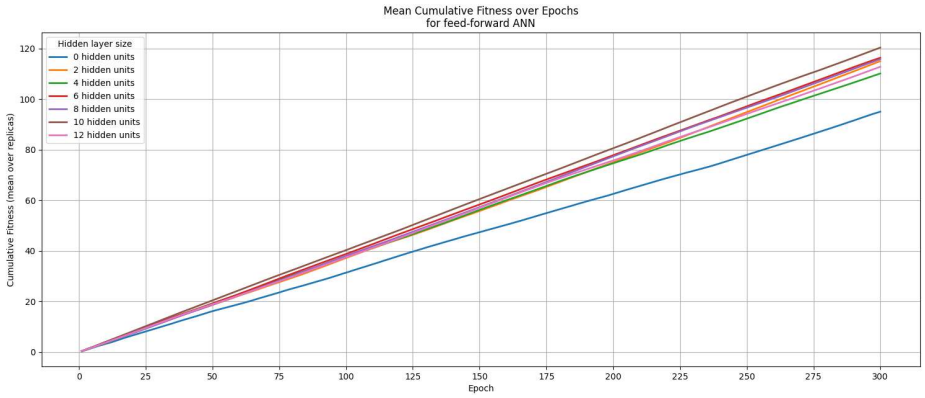

Figure 3.5: Mean cumulative fitness across epochs for continuous Feed Forward NN-based mechanism under no-noise condition.

In both cases, with and without noise, $H = 10$ outperforms the others, reaching even higher values in the noisy setting. It is followed by $H = 8$ and $H = 6$ variants, which perform almost identically and consistently rank just below the best, in the noiseless case. While under noisy conditions, $H = 12$ takes the second position,

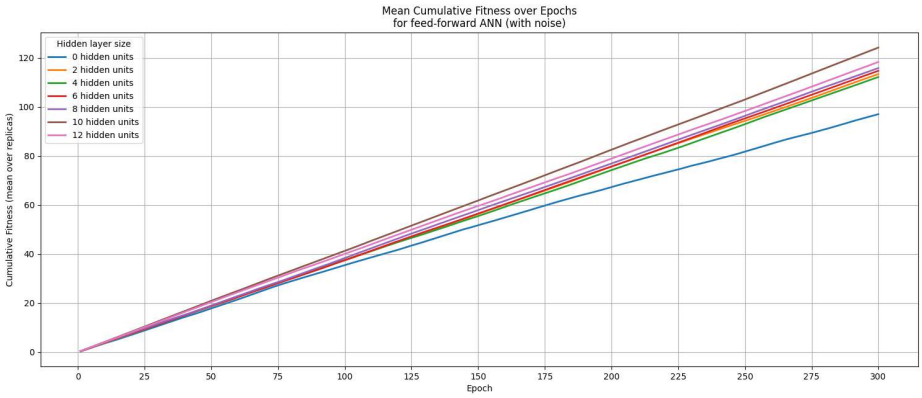

Figure 3.6: Mean cumulative fitness across epochs for continuous Feed Forward NN-based mechanism under noise condition.

overtaking both $H = 6$ and $H = 8$. Despite this result, $H = 12$ exhibits an average-to-low performance in the noiseless case.

When there is no noise, $H = 2$ starts slower but narrows the gap by the end of the run, approaching the performance of $H = 6$ and $H = 8$, though the separation between these curves is more visible between epochs 200 and 250. In the noisy case, $H = 2$ is in the middle performing groups with $H = 4$, $H = 6$, and $H = 8$, which all perform very similarly.

The $H = 0$ (no hidden layer) case remains the worst-performing overall, significantly behind when running either with or without noise.

Final Fitness Sum Distribution The distribution of final cumulative fitness for the Feed-Forward NN with various hidden layer sizes is depicted in Figure 3.7 and Figure 3.8.

The version with $H = 10$ hidden units achieves the highest median fitness, under both noiseless and noisy conditions. In the latter case, its performance improves, showing a higher median and a tighter IQR and whiskers, indicating better stability.

$H = 6$ conquers second place when run without noise, achieving a slightly lower median than the best one, but a more compact box plot spread, suggesting a more consistent behavior. Under noisy conditions, $H = 12$ exhibits the second-highest median fitness, while $H = 6$ follows closely with a marginally lower median but a

broader IQR and overall distribution. In the opposite scenario, $H = 12$ retains a relatively low median and a wide spread.

Other configurations, $H = 2$, $H = 4$, and $H = 8$, showcase similar achievements in both settings, while $H = 0$ once again performs the worst, characterized by the lowest median.

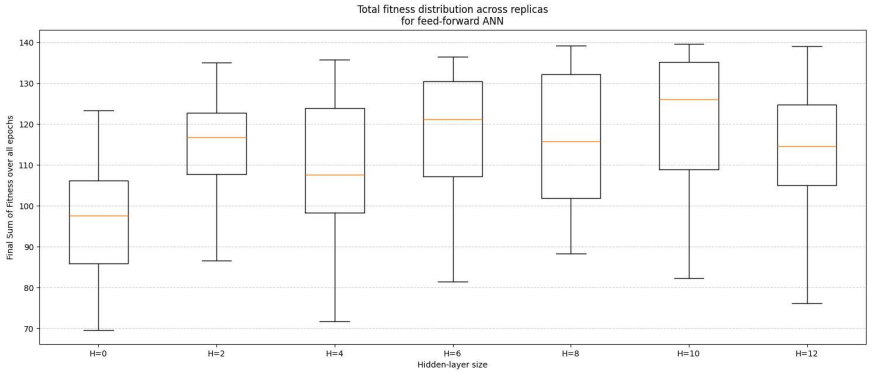

Figure 3.7: Distribution of total fitness across replicas under no-noise condition (Feed Forward NN-based mechanism).

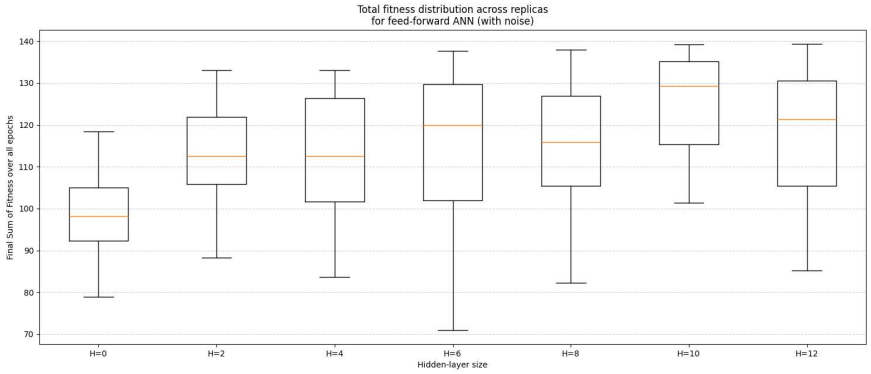

Figure 3.8: Distribution of total fitness across replicas under noise condition (Feed Forward NN-based mechanism).

Maximum Fitness Distribution Figure 3.9 and Figure 3.10 illustrate the distribution of maximum fitness values across replicas for each H configuration.

In the noiseless setting, the controller with $H = 6$ hidden units slightly outperforms all other configurations, closely followed by $H = 2$, which also has the

smallest IQR, and $H = 10$. In this setting, the worst-performing variants are $H = 4$ and $H = 12$, as evidenced by their lower medians and broader spreads, reflecting both reduced peak performance and greater variability.

Under noisy conditions, the $H = 10$ configuration emerges as the top performer, with the highest median maximum fitness and a compact box plot spread, suggesting both robustness and high peak capability. $H = 6$, $H = 8$, and $H = 12$ share the second spot with similar median values. Among them, $H = 6$ has the narrowest IQR, reinforcing its stability, whereas $H = 8$ exhibits one of the widest ones, indicating more erratic behavior. In this case, $H = 0$, $H = 2$, and $H = 4$ present the lowest medians.

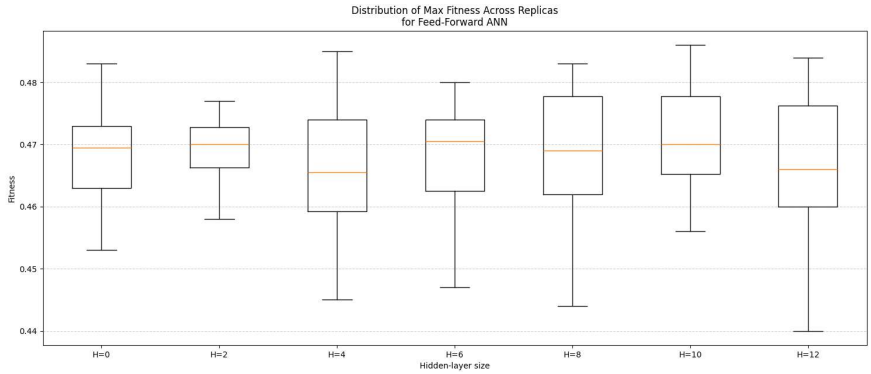

Figure 3.9: Distribution of maximum fitness across replicas under no-noise condition (Feed Forward NN-based mechanism).

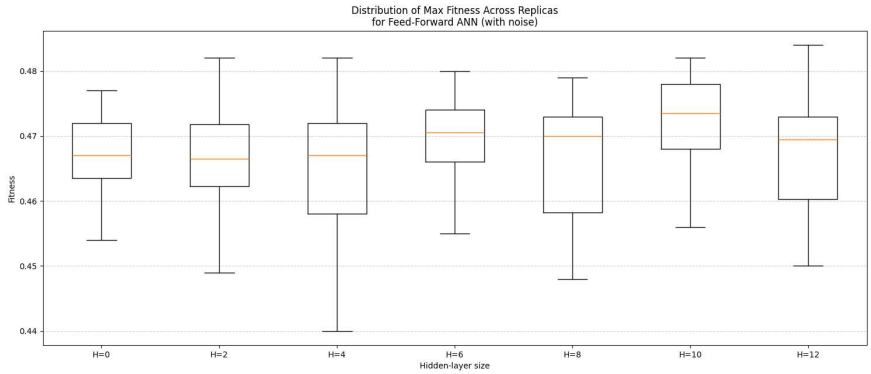

Figure 3.10: Distribution of maximum fitness across replicas under noise condition (Feed Forward NN-based mechanism).

Cumulative % of Successful Replicas The last metric used provides insights on the cumulative % of replicas that surpass a certain fitness value threshold, as can be observed in Figure 3.5 and Figure 3.6.

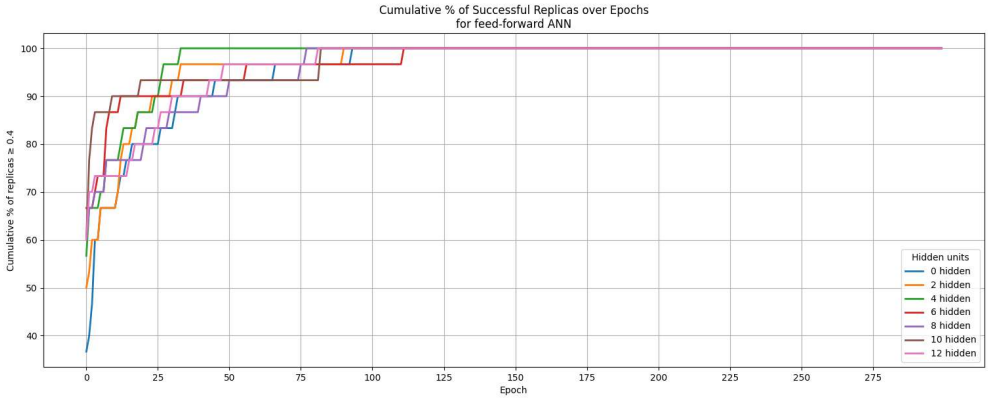

Figure 3.11: Cumulative percentage of successful replicas over time under no-noise condition (Feed Forward NN-based mechanism).

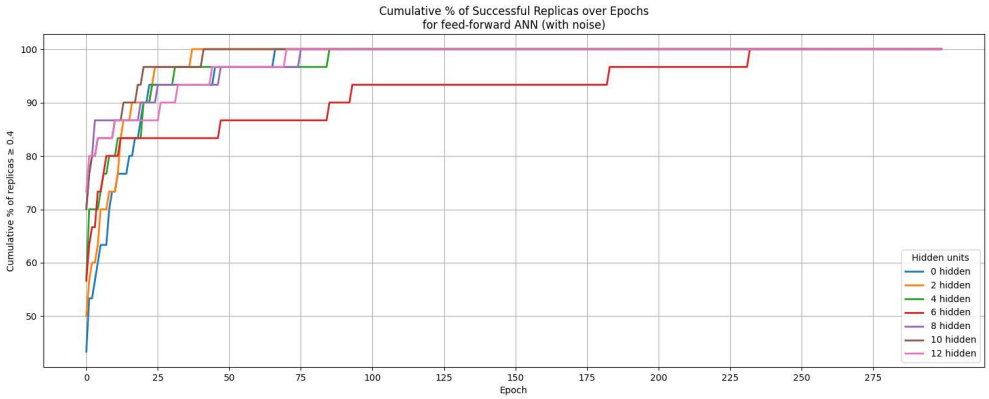

Figure 3.12: Cumulative percentage of successful replicas over time under noise condition (Feed Forward NN-based mechanism).

In the noiseless condition, the configuration with $H = 4$ is the first to reach 100% successful replicas, well before the others. $H = 0$, $H = 2$, $H = 8$, $H = 10$, and $H = 12$ all reach 100% around one third of total epochs. The slowest to reach full success is $H = 6$, despite beginning with the highest initial percentage. Interestingly, while $H = 10$ is not the earliest to saturate, it shows the steepest curve in the earliest epochs, indicating fast initial adaptation.

In the noisy condition, the $H = 2$ controller is the first to reach 100% successful replicas, closely followed by $H = 10$. Most other configurations ($H = 0$, $H = 4$, $H = 8$, and $H = 12$) behave similarly, reaching full success before one-third of the execution epochs. The $H = 6$ configuration, which had the slowest convergence even in the noiseless case, again falls behind all others, only reaching full success during the last third of the epochs. In the very early epochs, the steepest initial gains are observed for $H = 4$, $H = 8$, and $H = 12$, indicating rapid initial growth, with $H = 12$ also starting with the highest percentage of replicas above threshold.

3.2 Recurrent Neural Network

This mechanism extends the architecture of the Feed-Forward Neural Network by incorporating a memory component into the network. The main idea remains unchanged: the robot must explore the environment and, over successive epochs, optimize its behavior to avoid obstacles. However, while in the feed-forward approach each decision depends solely on current sensor readings, in the recurrent mechanism each neuron also integrates past internal states. The core algorithm is described in Pseudocode 9.

Algorithm 9: Online adaptation for Recurrent NN mechanism

```
θ ← vector of PARAMS random real-valued weights ∈ [−1, 1]
best_θ ← θ
best_fitness ← −∞
prev_hidden ← H real-valued units initialized to 0
prev_output ← 2 real-valued units initialized to 0
for epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for step in 1 to MOVE_STEPS do
        sense environment → (s_L, s_R)
        (outL, outR) ← RNN_forward(s_L, s_R, θ, prev_hidden, prev_output)
        set motor speeds using output
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum / MOVE_STEPS
    if avg_fitness > best_fitness then
        best_θ ← θ
        best_fitness ← avg_fitness
    end
    else
        θ ← mutate(θ)
    end
end
```

For each epoch, the robot's behavior is based on a fixed configuration θ, which encodes the neural network's weights and biases. During each step of an epoch, sensor readings are processed by the function described in Algorithm 10, which computes the motor outputs based on the current configuration θ. When no hidden

layer is present, recurrence is handled directly through feedback from the previous output values. When a hidden layer is included, recurrent connections are applied to hidden activations, which then influence the final output computation.

Algorithm 10: `RNN_forward`: Recurrent Neural Network activation with memory

Function `RNN_forward`(s_L, s_R)**:**
  if $H = 0$ **then**
    $z_L \leftarrow w_{L1} \cdot s_L + w_{L2} \cdot s_R + b_L + \alpha_L \cdot \texttt{prev\_output}[1]$
    $z_R \leftarrow w_{R1} \cdot s_L + w_{R2} \cdot s_R + b_R + \alpha_R \cdot \texttt{prev\_output}[2]$
    `outL` $\leftarrow \texttt{sigmoid}(z_L)$
    `outR` $\leftarrow \texttt{sigmoid}(z_R)$
    `prev_output` $\leftarrow$ (`outL`, `outR`)
  end
  else
    foreach *hidden neuron* $j = 1$ *to* H **do**
      if j *is odd* **then**
        $\bar{j} \leftarrow j + 1$
      else
        $\bar{j} \leftarrow j - 1$
      end
      $z \leftarrow w_{jL} \cdot s_L + w_{jR} \cdot s_R + b_j$
          $+\alpha_{\texttt{self}} \cdot \texttt{prev\_hidden}[j] + \alpha_{\texttt{cross}} \cdot \texttt{prev\_hidden}[\bar{j}]$
      $h_j \leftarrow \texttt{sigmoid}(z)$
    end
    $z_L \leftarrow \sum_j w_{Lj} \cdot h_j + b_L$
    $z_R \leftarrow \sum_j w_{Rj} \cdot h_j + b_R$
    `outL` $\leftarrow \texttt{sigmoid}(z_L)$
    `outR` $\leftarrow \texttt{sigmoid}(z_R)$
    `prev_hidden` $\leftarrow (h_1, h_2, \ldots, h_H)$
    `prev_output` $\leftarrow$ (`outL`, `outR`)
  end
  return (`outL`, `outR`)

A behavior (resulting from a configuration θ) is used and evaluated throughout an entire epoch; at each step, its produced fitness is calculated using the same function used for the previous mechanisms (see 2). The fitness values are accumulated and then checked at the end of the epoch; if the configuration θ has led to

better performance than the current best, it is stored as the new best. Otherwise, a single parameter is perturbed via the same mutation strategy adopted for the Feed-Forward NN, explained in Algorithm 7.

The structure of the network and the way the parameters are interpreted differ depending on whether we include a hidden layer or not:

- **No hidden layer ($H = 0$):** in the architecture, illustrated in Figure 3.13, each output receives weighted input from both proximity sensors, plus a bias term and a recurrent loop from its own previous output (obtained in the preceding step). In this case, the total number of used parameters is:

$$\texttt{PARAMS} = I \cdot O + O + O$$

  where $I = 2$ inputs (left/right sensor), $O = 2$ outputs (left/right wheels). The first O terms represent the input-to-output weights, the next O are output biases, and the final O are the recurrent self-loop weights for each output neuron (denoted as α_L, α_R).

- **With hidden layer ($H > 0$):** the network, illustrated in Figure 3.14, includes H hidden neurons, and recurrence is applied at the hidden layer. Each hidden neuron receives input from both sensors, a bias, a recurrent self-loop, and a cross-loop from a paired hidden neuron $\bar{j}$. The total number of parameters in this case is:

$$\texttt{PARAM\_COUNT} = I \cdot H + H + H \cdot 2 + H \cdot O + O$$

  Where the terms respectively account for: input-to-hidden weights, hidden biases, loop and cross-loop recurrent weights per hidden neuron, hidden-to-output weights, and output biases. Each hidden neuron has two recurrent terms: one self-loop (α_{jj}), and one cross-loop from a paired neuron ($\alpha_{j\bar{j}}$), where:

$$\bar{j} = \begin{cases} j+1, & \text{if } j \text{ is odd} \\ j-1, & \text{if } j \text{ is even} \end{cases}$$

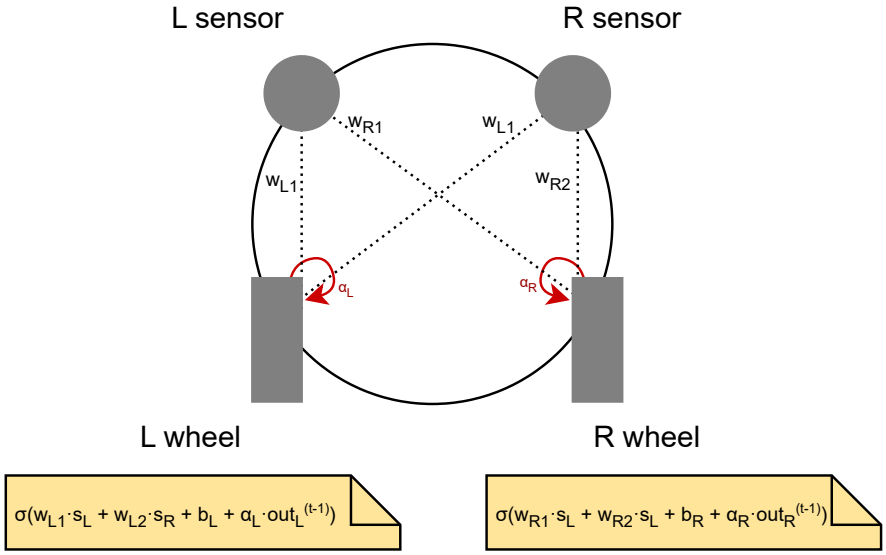

Figure 3.13: Schematic representation of the RNN architecture with no hidden layer. Each wheel is controlled by a single output neuron receiving input from both proximity sensors, a bias term, and a recurrent self-loop contribution from its previous output ($\alpha_L \cdot out_L^{(t-1)}$ for the left wheel, $\alpha_R \cdot out_R^{(t-1)}$ for the right wheel).

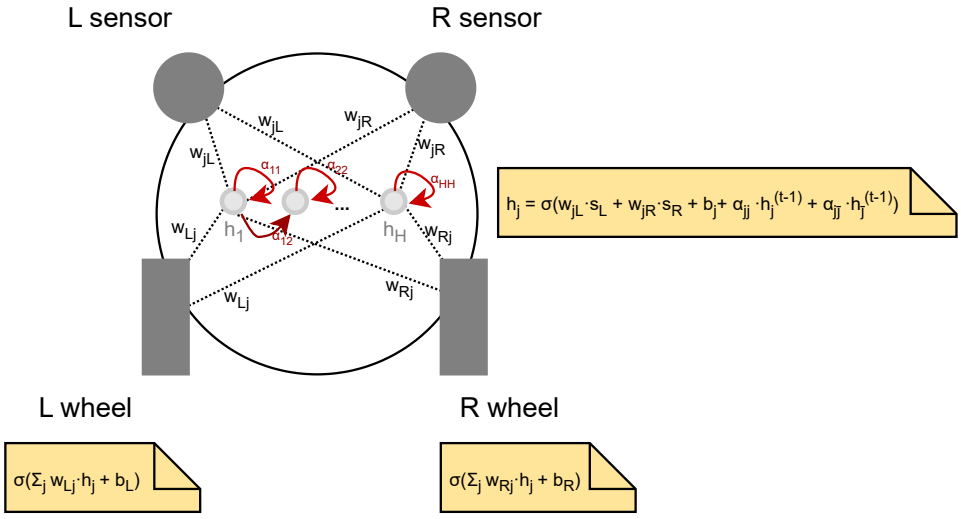

Figure 3.14: Schematic representation of the general RNN architecture with a hidden layer. Each hidden neuron h_j receives weighted input from both sensors, a bias term b_j, a recurrent self-connection ($\alpha_{jj} \cdot h_j^{(t-1)}$), and a cross-connection from a paired hidden unit ($\alpha_{j\bar{j}} \cdot h_{\bar{j}}^{(t-1)}$). $\bar{j}$ denotes the partner neuron of j.

Similarly to previously mentioned cases, this mechanism was tested in the same arena (see Figure 2.4), using different hidden layer (H) sizes. For each hidden layer size tested, a total of 60 independent runs were executed; half of these added uniform additive noise (± 0.01) to proximity sensors and wheel actuators. For each epoch of each run, we saved the epoch number with the corresponding fitness value produced. The acquired data was examined using the same suite of metrics as the Feed Forward NN.

Average Fitness Over Time Figure 3.15 and Figure 3.16 show the evolution of average fitness for different hidden layer sizes H, in both noiseless and noisy conditions.

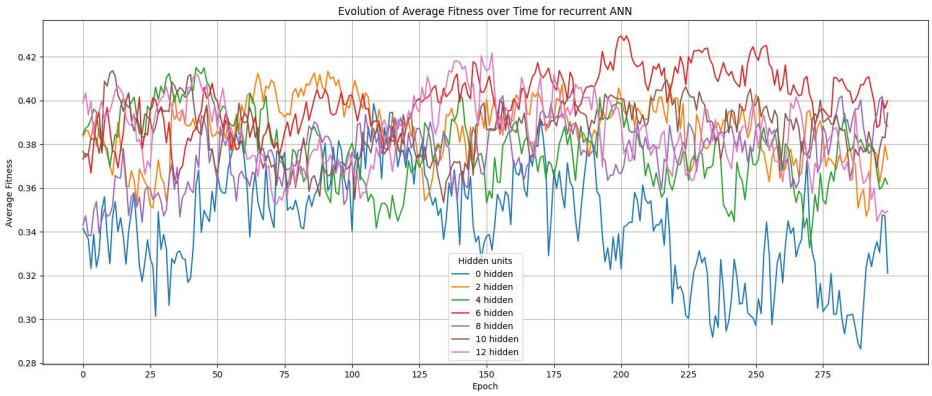

Figure 3.15: Evolution of average fitness across epochs for Recurrent NN-based mechanism under no-noise condition, with different hidden layer sizes.

In both scenarios, the network without a hidden layer ($H = 0$) underperforms compared to all other configurations; its average fitness consistently falls below the others and exhibits high instability.

Among all other variants with $H > 0$, performance is similar both in noiseless and noisy conditions. In the first setting, $H = 6$ reaches the highest values on the last epochs, while maintaining relatively high values throughout the run. Other hidden layer sizes also perform well and achieve high values (e.g., $H = 10$, $H = 12$), but they show more instability. When noise is introduced, the distinction between architectures becomes less sharp. In the second half of the epochs, all versions except $H = 0$ tend to converge toward similar average fitness ranges, with the

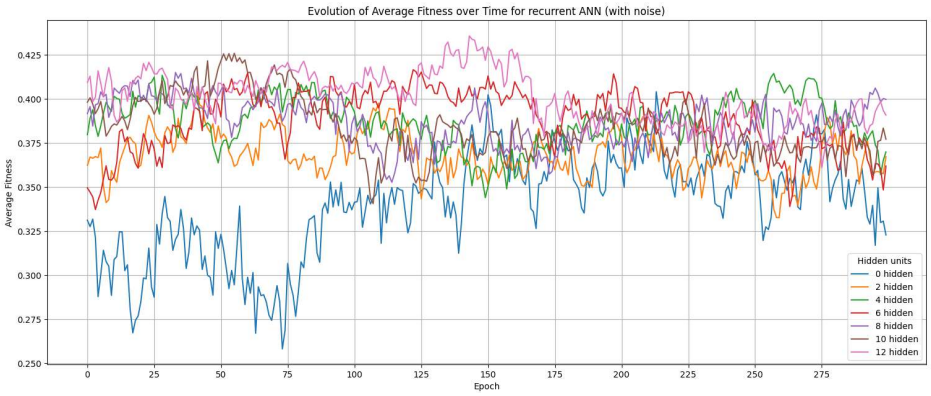

Figure 3.16: Evolution of average fitness across epochs for Recurrent NN-based mechanism under noise condition, with different hidden layer sizes.

highest peaks obtained by $H = 10$ and $H = 12$. Although $H = 6$ also performs well, it suffers a drop in the final epochs, while $H = 12$ maintains a more stable trend overall.

Compared to the feed-forward network, the recurrent architecture appears slightly more resilient to noise and maintains a tighter grouping of fitness across hidden sizes. Moreover, while the feed-forward NN favored larger hidden layers (with $H = 10$ dominating, see Section 3.1), the RNN performs well even with smaller hidden sizes, such as $H = 6$.

Mean Cumulative Fitness From Figure 3.17 and Figure 3.18, we can observe the evolution of cumulative fitness across epochs for each network size under both noiseless and noisy conditions.

Once again, $H = 0$ has the worst performance, consistently accumulating the least fitness over time, regardless of noise condition. In the noiseless setting, $H = 6$ achieves the best cumulative performance, clearly dominating all other curves after one third of the epochs. It is closely followed by $H = 2$, $H = 10$, and $H = 12$, which perform almost identically. In the presence of noise, the network with $H = 12$ surpasses all others from the beginning, consistently accumulating the highest total fitness throughout the run. It is followed by a dense group composed of $H = 4$, $H = 6$, $H = 8$, and $H = 10$, which all perform comparably, with overlapping curves. $H = 2$ joins $H = 0$ in the lowest performing group, falling behind all other

configurations after ≈ one-fourth of epochs.

Compared to the Feed-Forward NN (see Section 3.1), the recurrent version delivers a more robust and tighter distribution of performance across hidden layer sizes. In both networks, the inclusion of hidden units substantially improves cumulative performance, with the highest cumulative fitness values being ≈ 120.

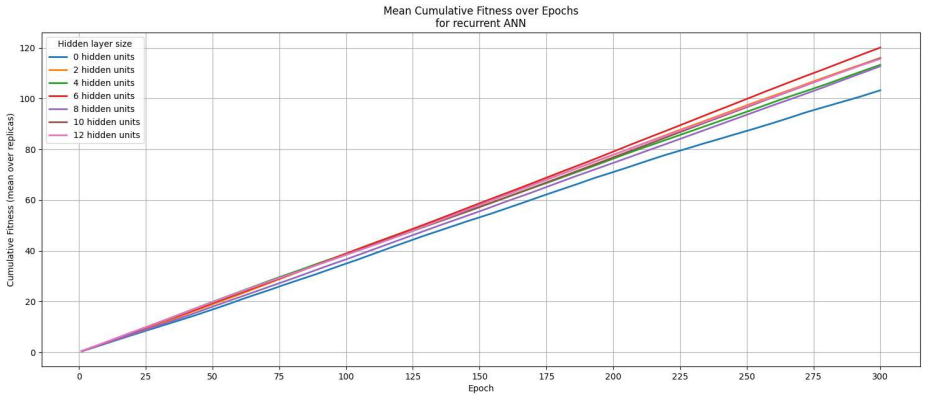

Figure 3.17: Mean cumulative fitness across epochs for continuous Recurrent NN-based mechanism under no-noise condition.

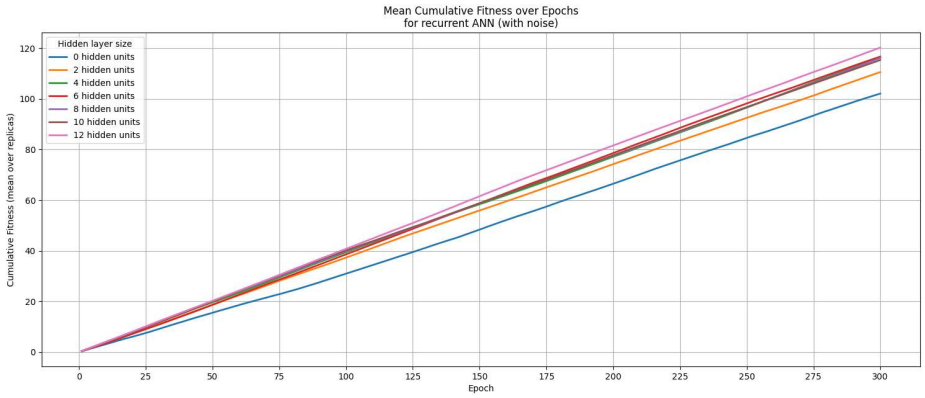

Figure 3.18: Mean cumulative fitness across epochs for continuous Recurrent NN-based mechanism under noise condition.

Final Fitness Sum Distribution Figure 3.19 and Figure 3.20 illustrate the distribution of total fitness values accumulated across all epochs for each hidden layer size in the recurrent NN.

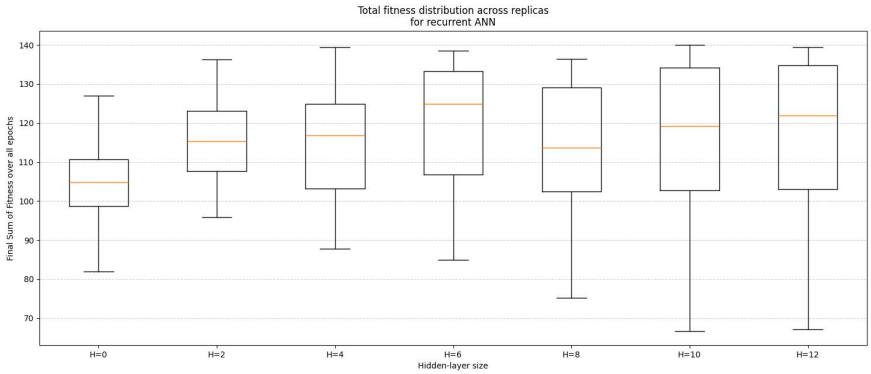

Figure 3.19: Distribution of total fitness across replicas under no-noise condition (Recurrent NN-based mechanism).

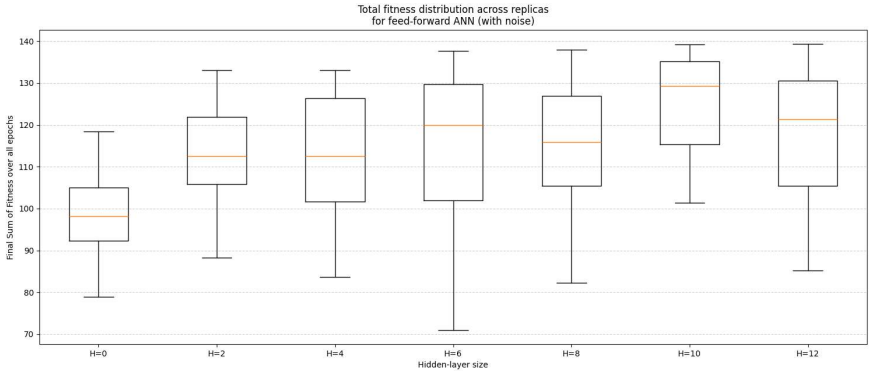

Figure 3.20: Distribution of total fitness across replicas under noise condition (Recurrent NN-based mechanism).

In the noiseless scenario, the configuration with $H = 6$ hidden units emerges as the best-performing variant, achieving the highest median value and a compact box plot spread. While $H = 10$ and $H = 12$ reach similarly high medians, their spreads are wider, especially in the lower whiskers, suggesting more instability in outcomes.

Under noise, both $H = 6$ and $H = 12$ tie for the highest median fitness, with similar box shapes, indicating robust behavior even under perturbation. Their medians are closely followed by $H = 10$, though $H = 10$ now exhibits the widest IQR of all, suggesting inconsistent results.

As in previous evaluations, the $H = 0$ variant performs the worst overall, with the lowest median fitness and a tight but lower distribution spread.

Maximum Fitness Distribution The distribution of the maximum fitness values obtained from the Recurrent NN is shown in Figure 3.21 and Figure 3.22.

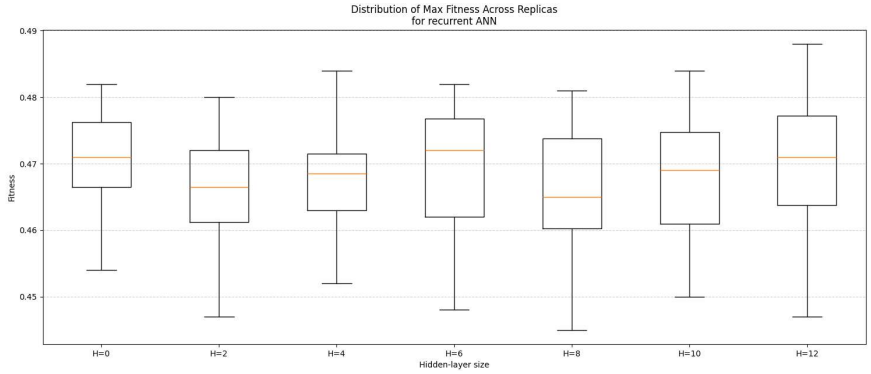

Figure 3.21: Distribution of maximum fitness across replicas under no-noise condition (Recurrent NN-based mechanism).

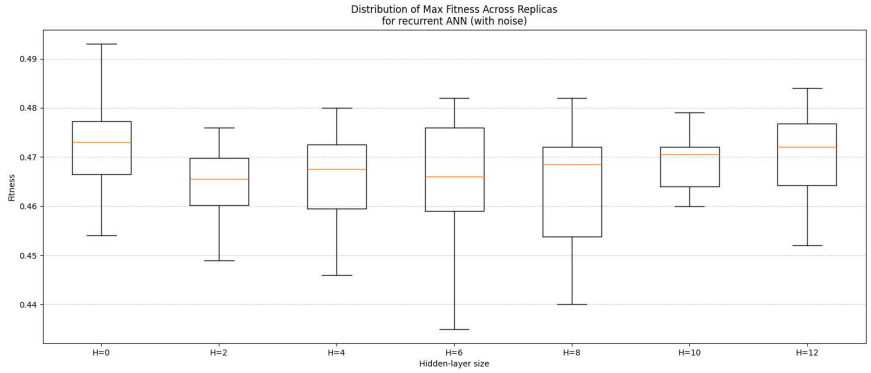

Figure 3.22: Distribution of maximum fitness across replicas under noise condition (Recurrent NN-based mechanism).

In the noiseless setting, the $H = 6$ controller emerges as the top performer, achieving the highest median maximum fitness. Surprisingly, $H = 0$ and $H = 12$ achieve the second-best median and tighter IQRs. However, $H = 12$ also presents the widest whiskers, indicating a broader variance in peak performance across runs. In this case, the lowest median was obtained by $H = 8$.

When tested with noise, $H = 0$ interestingly reaches the highest median alongside $H = 12$. Both maintain relatively tight inter-quartile ranges, even if $H = 0$ presents a very wide upper whisker. $H = 10$ secures both the second-highest me-

dian and the smallest IQR and whiskers among all, indicating excellent robustness even with noise disturbance. The lowest median values are brought by $H = 2$ and $H = 6$.

Cumulative % of Successful Replicas Figure 3.23 and Figure 3.24 show the evolution of the cumulative percentage of successful replicas, defined as those reaching average fitness ≥ 0.4 over training epochs, for various hidden layer sizes H, both in noiseless and noisy conditions.

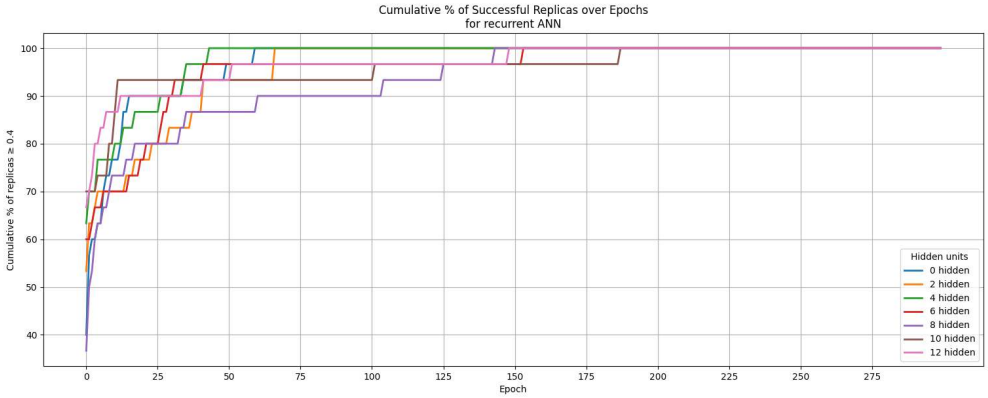

Figure 3.23: Cumulative percentage of successful replicas over time under no-noise condition (Recurrent NN-based mechanism).

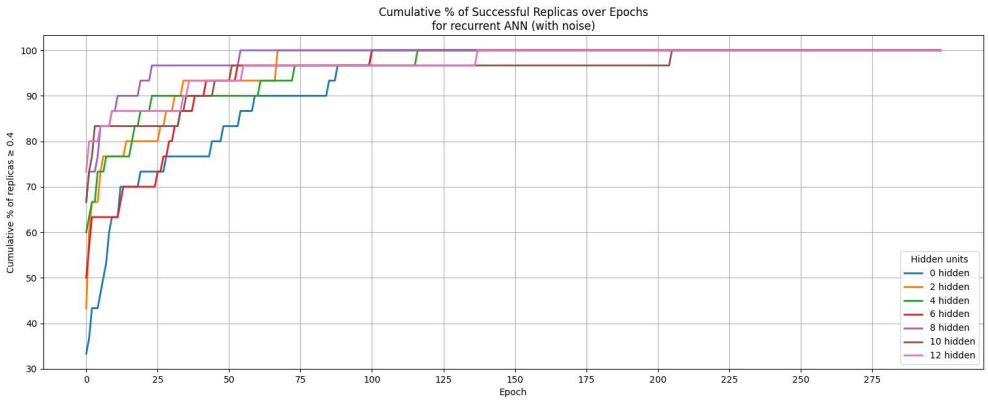

Figure 3.24: Cumulative percentage of successful replicas over time under noise condition (Recurrent NN-based mechanism).

In the noiseless scenario, $H = 4$ is the first to record 100% successful replicas, converging rapidly within the first one-sixth epochs. It is followed by $H = 0$ and $H = 2$, which also reach complete success closely after. Interestingly, while $H = 10$ is the last to reach 100%, it starts with the highest initial success rate, surpassing 90% before the others. Meanwhile, $H = 6$, $H = 8$, and $H = 12$ exhibit similar behaviors, all reaching full success midway.

In the noisy condition, the dynamics shift. The first configuration to reach 100% is $H = 8$, which demonstrates the steepest growth during the initial 50 epochs. It is shortly followed by $H = 2$, which once again shows strong performance. The final configuration to converge is again $H = 10$, reaching 100% around epoch 200, maintaining its trend of fast early learning but delayed full success. Notably, $H = 12$ begins with the highest initial success rate. Overall, all hidden-layer configurations except $H = 10$ show convergence within the first half of the training run, demonstrating robustness even in noisy environments.

3.3 RNN variants

This section is dedicated to all the variants created and tested based on the RNN mechanism discussed earlier.

While both feed-forward and recurrent architectures demonstrated comparable performance trends across metrics, the Recurrent NN was selected for further experimentation not only due to its architectural interest, but also because it consistently achieved higher cumulative fitness, more stable maximum fitness across replicas, and faster convergence to successful behaviors, even when in the presence of noise. Accordingly, all subsequent experiments were conducted in the presence of additive noise (± 0.01), applied to both proximity sensor readings and wheel actuators.

All further presented experiments were tested with a fixed layer size of $H = 10$. $H = 10$ was chosen as a balanced compromise between robustness and early-stage effectiveness. To carry out this choice, we only considered *Cumulative Fitness Sum distribution*, *Maximum Fitness distribution*, and *Cumulative % of successful replicas*; prioritized in this order.

In the final fitness sum distribution box plot (see Figure 3.20), the top three

configurations in terms of median performance are $H = 12$, $H = 6$, and $H = 10$. In the maximum fitness distribution (Figure 3.22), $H = 10$ again exhibits strong results. It records the third-highest median (just behind $H = 0$ and $H = 12$), but it outperforms both in stability, presenting the tightest IQR and whiskers of all configurations. Lastly, the cumulative percentage of successful replicas plot (Figure 3.18) shows that $H = 10$ has the fastest growth in the very early stages, being the first to surpass 90% of successful replicas. Although $H = 6$, $H = 8$, and $H = 12$ eventually reach 100% before, the early dominance of $H = 10$ reflects a strong initial learning curve, which can be valuable when adaptation speed is critical. All these considerations make $H = 10$ a reliable, robust, and balanced choice for the subsequent experiments.

3.3.1 Exploration VS Exploitation variants

In online adaptation, a mechanism must repeatedly decide whether to exploit the current configuration θ or explore new ones through mutation. This decision is known as the *exploration–exploitation trade-off*, a foundational dilemma in reinforcement learning and evolutionary strategies [SB18]. On one hand, exploitation favors the best-known behavior to maximize short-term reward; on the other, exploration introduces random variation that may uncover better solutions in the long term, especially in non-stationary settings or environments that aren't fully known. It's crucial to find the right balance between the two: too much exploitation risks premature convergence to suboptimal configurations, while excessive exploration may prevent convergence altogether.

This section presents thirteen different RNN mechanisms variants, which aim to further explore this trade-off in online learning. The structure of these mechanisms follows a similar adaptation loop, described in the pseudocode at 11, however they can be differentiated and grouped based on whether and how they incorporate memory of past configurations, and how they balance exploration (i.e., mutating the current configuration θ) with exploitation (i.e., retaining a previously effective configuration).

In the first group of strategies, the system does not retain any memory of previously used configurations. Still, its only available knowledge, at the end of

Algorithm 11: Online adaptation loop for exploration–exploitation variants

```
θ ← random vector of neural weights
past_θ ← either none, previous or all
for epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for step in 1 to MOVE_STEPS do
        sense environment → (s_L, s_R)
        (outL, outR) ← RNN_forward(s_L, s_R, θ)
        set motor speeds from output
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum/MOVE_STEPS
    configuration_adaptation(avg_fitness, epoch)
end
```

each epoch, is the current fitness. Four sub-variants were implemented:

- **No Memory 50–50:** a random value $p \in [0, 1]$ is drawn at the end of each epoch. If $p \in [0.5, 1]$, the system explores by mutating θ; if $p \in [0, 0.5)$, it exploits by retaining the current configuration, the only configuration it knows.

- **No Memory 10–90:** if $p \in [0.1, 1]$, the system explores; if $p \in [0, 0.1)$, it retains θ.

- **No Memory 90–10:** if $p \in [0.9, 1]$, the system explores via mutation; if $p \in [0, 0.9)$, it keeps the current configuration.

- **No Memory Epoch-Based:** an exploration threshold

$$p_{\text{explore}} = 1 - \frac{\texttt{epoch}}{\texttt{MAX\_EPOCHS}} \tag{3.1}$$

  is computed dynamically. If $p \in [0, p_{\text{explore}}]$, exploration is triggered; otherwise ($p \in (p_{\text{explore}}, 1]$), the system exploits.

The logic of these variants is formalized in Algorithm 12.

Algorithm 12: Exploration vs. Exploitation - no past configuration memory

```
Function configuration_adaptation(avg_fitness, epoch):
    p ← random uniform value in [0, 1]
    if variant is no memory 50–50 then
        if p ≥ 0.5 then
            θ ← mutate(θ)
        end
        else
            retain current θ
        end
    else if variant is no memory 10–90 then
        if p ≥ 0.1 then
            θ ← mutate(θ)
        end
        else
            retain current θ
        end
    else if variant is no memory 90–10 then
        if p > 0.9 then
            θ ← mutate(θ)
        end
        else
            retain current θ
        end
    else if variant is no memory epoch-based then
        p_explore ← 1 − epoch / MAX_EPOCHS
        if p ≤ p_explore then
            θ ← mutate(θ)
        end
        else
            retain current θ
        end
```

The second group of variants compares the current configuration to the one used in the previous epoch. After computing average fitness, the system decides either to mutate the current θ or exploit the fitter configuration between the two. Again, four different sub-implementations are considered:

- **Prev vs Current 50–50:** a random value $p \in [0, 1]$ is sampled. If $p \in [0, 0.5)$, the system exploits by retaining the fitter configuration between the one used in the previous epoch and the current one. If $p \in [0.5, 1]$, exploration occurs via mutation of θ.

- **Prev vs Current 10–90:** if $p \in [0, 0.1)$, the system selects the fitter of the current and previous configurations; if $p \in [0.1, 1]$, it mutates θ.

- **Prev vs Current 90–10:** if $p \in [0, 0.9]$, the fitter of the two recent configurations is preserved; if $p \in (0.9, 1]$, mutation is triggered.

- **Prev vs Current Epoch-Based:** an epoch-based exploration probability is computed exactly as for the previously mentioned group (see Equation (3.1)). If $p \in [0, p_{\text{explore}}]$, exploration is performed (mutation); otherwise ($p \in (p_{\text{explore}}, 1]$), exploitation selects the best between current and previous θ.

This family of mechanisms is described in Algorithm 13.

For the third group, rather than comparing only to the immediately previous configuration, these strategies track and retain all previously evaluated configurations, selecting the all-time best one (either the current θ or from any of the past ones). Even in this case, we have four analogous sub-mechanisms:

- **All vs Current 50–50:** at the end of each epoch, a value $p \in [0, 1]$ is sampled. If $p \in [0, 0.5)$, the best configuration is selected between the current one and the best seen so far across all epochs (exploitation); if $p \in [0.5, 1]$, mutation is applied (exploration).

- **All vs Current 10–90:** if $p \in [0, 0.1)$, exploitation occurs by reusing the best configuration overall; otherwise ($p \in [0.1, 1]$), the system explores via mutation.

- **All vs Current 90–10:** with $p \in [0, 0.9]$, the algorithm exploits by using the best-so-far configuration; if $p \in (0.9, 1]$, exploration via mutation is performed.

Algorithm 13: Exploitation vs Exploration - previous fitness memory

Function *configuration_adaptation(avg_fitness, prev_fitness, epoch)*:
  $p \leftarrow$ `random uniform in` $[0, 1]$
  if *variant is* ***prev vs current 50–50*** **then**
    if $p < 0.5$ **then**
      Use the fitter of the current (epoch e) or previous (epoch e-1) configuration.
    end
    else
      $\theta \leftarrow$ `mutate`(θ)
    end
  else if *variant is* ***prev vs current 10–90*** **then**
    if $p < 0.1$ **then**
      ... same logic as above ...
    end
    else
      $\theta \leftarrow$ `mutate`(θ)
    end
  else if *variant is* ***prev vs current 90–10*** **then**
    if $p \leq 0.9$ **then**
      ... same logic as above ...
    end
    else
      $\theta \leftarrow$ `mutate`(θ)
    end
  else if *variant is* ***prev vs current epoch-based*** **then**
    $p_{\text{explore}} \leftarrow 1 - \frac{\texttt{epoch}}{\texttt{MAX\_EPOCHS}}$
    if $p \leq p_{explore}$ **then**
      $\theta \leftarrow$ `mutate`(θ)
    end
    else
      Use the fitter of the current (epoch e) or previous (epoch e-1) configuration.
    end

- **All vs Current Epoch-Based:** the exploration probability is computed using the formula in Equation (3.1)). If $p \in [0, p_{\text{explore}}]$, the current configuration is mutated; otherwise ($p \in (p_{\text{explore}}, 1]$), the best among all past and

current configurations is selected for the next epoch.

All approaches from the third group are formally described in Algorithm 14.

Algorithm 14: Configuration adaptation - full memory

Function *configuration_adaptation(avg_fitness, epoch)*:
  $p \leftarrow$ `random uniform in` $[0, 1]$
  best_so_far $\leftarrow$ configuration with highest fitness seen so far
  if *variant is* ***all vs current 50–50*** **then**
    if $p < 0.5$ **then**
      Use the fitter of the current or best-so-far configuration.
    end
    else
      $\theta \leftarrow$ `mutate`(θ)
    end
  else if *variant is* ***all vs current 10–90*** **then**
    if $p < 0.1$ **then**
      ... same logic as above ...
    end
    else
      $\theta \leftarrow$ `mutate`(θ)
    end
  else if *variant is* ***all vs current 90–10*** **then**
    if $p \leq 0.9$ **then**
      ... same logic as above ...
    end
    else
      $\theta \leftarrow$ `mutate`(θ)
    end
  else if *variant is* ***all vs current epoch-based*** **then**
    $p_{\text{explore}} \leftarrow 1 - \frac{\texttt{epoch}}{\texttt{MAX\_EPOCHS}}$
    if $p \leq p_{explore}$ **then**
      $\theta \leftarrow$ `mutate`(θ)
    end
    else
      Use the fitter of the current or best-so-far configuration.
    end

Lastly, in this final mechanism, the exploration probability is dynamically adapted based on both epoch progression and the variance in recent fitness perfor-

mance. A sliding window of the last W average fitness values is maintained and used to compute a variance score v. If variance is low, possibly indicating stagnation, exploration is boosted by an additional factor scaled to the current epoch. Additionally, regardless of variance, if a performance drop is detected from the previous epoch to the current epoch, exploration probability is slightly increased. The resulting exploration probability p_e determines whether to mutate θ or reuse the better of the current and previous. This behavior is detailed in Algorithm 15.

Algorithm 15: Fitness Variance–Driven exploration vs exploitation

$\theta \leftarrow$ random vector of neural weights
`fitness_history` $\leftarrow$ empty window of size `WINDOW_SIZE`
for *epoch* *in* 1 *to* *MAX_EPOCHS* **do**
    `fitness_accum` $\leftarrow 0$
    for *step* *in* 1 *to* *MOVE_STEPS* **do**
        sense environment $\rightarrow (s_L, s_R)$
        $(\texttt{outL}, \texttt{outR}) \leftarrow \texttt{RNN\_forward}(s_L, s_R, \theta)$
        set motor speeds from output
        `fitness_accum` $+=$ `evaluate_performance()`
    end
    `avg_fitness` $\leftarrow$ `fitness_accum`/`MOVE_STEPS`
    update `fitness_history` with `avg_fitness`
    compute variance v over `fitness_history`
    $p_e \leftarrow$ `BASE_EXPLORATION`
    if $v <$ *VARIANCE_THRESHOLD* **then**
        $p_e \leftarrow p_e + (1 - \frac{\texttt{epoch}}{\texttt{MAX\_EPOCHS}}) \cdot \texttt{ADDITIONAL\_EXPLORATION}$
    end
    if *avg_fitness ¡ prev_fitness* **then**
        $p_e \leftarrow p_e +$ `DROP_BONUS`
    end
    $p \leftarrow$ random value in $[0, 1]$
    if $p \leq p_e$ **then**
        mutate θ
    end
    else
        Use the fitter of the current or previous configuration.
    end
    `prev_fitness` $\leftarrow$ `avg_fitness`
end

This time, considering the number of total mechanisms to evaluate, each variant was tested and evaluated on 10 independent runs, using the same arena (see

Figure 2.4). As previously explained, all executions are set to have a fixed hidden layer size of $H = 10$, and uniform additive noise (± 0.01) for proximity sensors and wheel actuators.

Final Fitness Sum Distribution Figure 3.25 illustrates the distribution of total fitness values accumulated over all epochs for each of the thirteen adaptation mechanisms explained above.

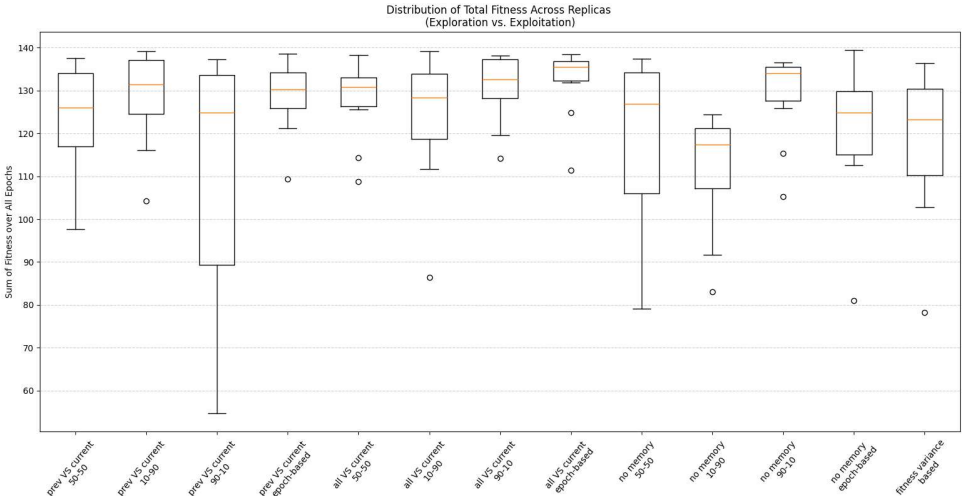

Figure 3.25: Distribution of total fitness across replicas (RNN Exploration VS Exploitation variants).

Among the memory-less variants, the 90–10 strategy stands out as the best-performing configuration, exhibiting the highest median and one of the narrowest IQRs, which translates to robust and consistent performance. While its opposite, the 10–90 variant, performs the worst, with the lowest median of the whole analysis, and a reasonably low whisker as well, indicating that excessive exploration in the absence of memory can impair convergence and stability. The 50–50 and epoch-based mechanisms lie in between: both achieve moderate and very similar medians, but the added stochasticity from their higher exploration rates leads to higher variability across runs, resulting in the second largest box plot spread.

In the *-1* memory group, retaining the precedent configuration, the 10–90 variant achieves the highest median among all configurations in this category, with a tight IQR and only a single mild outlier; aggressive exploration results prove

effective when paired with short-term memory. In contrast, the 90–10 variant underperforms significantly, showcasing the lowest median and widest IQR of all thirteen, possibly due to being stuck in suboptimal behaviors due to insufficient exploration. The balanced (50–50) approach yields average results, with a median similar to that of the memory-less $50 - 50$. Lastly, for this group, the epoch-based variant delivers a slightly lower median than the best of this group while presenting a smaller IQR and whiskers, offering a good compromise.

Regarding the third group (where the current configuration is tested against the best overall), the performances of these variants are similar to each other. The epoch-based mechanism is the clear winner overall; it obtains the highest median and the tightest spread across all other box plots. This demonstrates how combining stronger initial exploration with final exploitation of all-time bests is highly effective. The 90–10 variant also performs well, achieving a similarly high median and a quite narrow IQR. Despite the opposite 10–90 strategy being the worst in this group, it is still not among the worst overall. The fixed 50–50 variant concludes this group; it is again a middle-ground option, offering decent performance but not matching the stability or effectiveness of its epoch-based and 90–10 counterparts.

Ultimately, the fitness variance-based adaptation strategy yields an average performance. Its median isn't among the highest, but it remains competitively high. However, the IQR is moderately wide, and it does present the lowest outlier, suggesting high variability in outcomes.

Maximum Fitness Distribution The distribution of maximum fitness values is reported in Figure 3.26.

In the memory-less group, the 90–10 variant once again demonstrates strong performance, attaining the highest median in its group and one of the tightest IQRs overall. Its success suggests that, as a good solution is discovered, intense exploitation can deliver great results, even without past configuration memory. By contrast, the worst of this group and one of the worst overall is 10–90. Its median and upper bound are both limited, while the spread is relatively wider compared to many others in this analysis. The 50–50 strategy achieves a higher median and tighter IQR, showcasing better reliability. Meanwhile, the epoch-based variant

median is in between $50-50$ and $10-90$. However, it has a noticeably broader distribution.

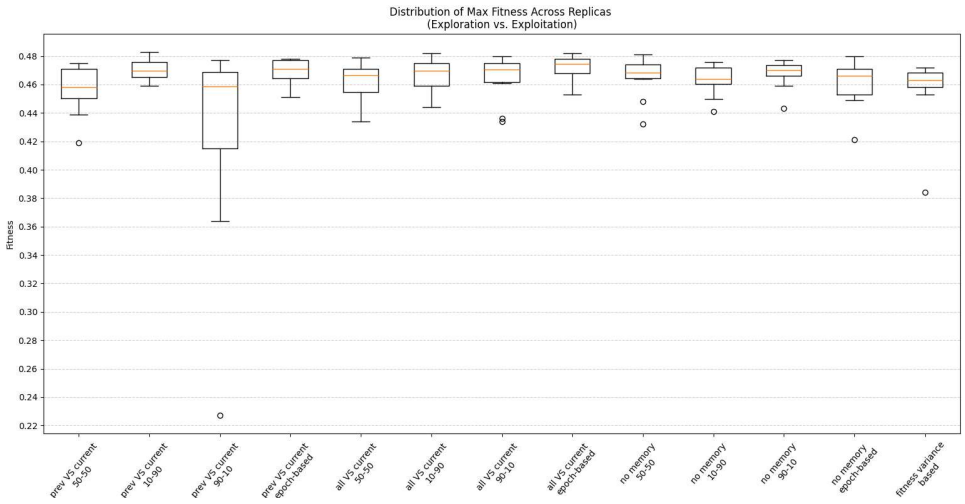

Figure 3.26: Distribution of maximum fitness across replicas (RNN Exploration VS Exploitation variants).

In the second group (*short-memory*), the 10–90 and epoch-based variants are among the strongest performers across all mechanisms, both achieving high medians and tight IQRs. Precisely, the epoch-based strategy slightly surpasses 10–90 in median. In contrast, the 90–10 variant exhibits very high variance, presenting the widest spread and a significantly low outlier too. Yet, the worst median value of all is achieved by the 50–50 strategy.

When considering all past configurations, the epoch-based variant dominates, achieving the highest median of all, confirming the advantage of combining full memory with decreasing exploration. Both the 90–10 and 10–90 variants follow closely; they have similar high medians, and while the second one shows a wider spread, the first one presents some outliers, bringing them to a tie. The 50–50 mechanism shows the least performing results within the group, but achieves an average overall performance.

The fitness variance-based mechanism had a moderate median, even if it does not achieve the absolute best fitness values. It stands out for its tight IQR, but a relatively low outlier suggests instability in outcomes.

Cumulative % of Successful Replicas As the final metric, Figure 3.27 tracks the cumulative number of replicas that reached a fitness score of at least 0.4 throughout training.

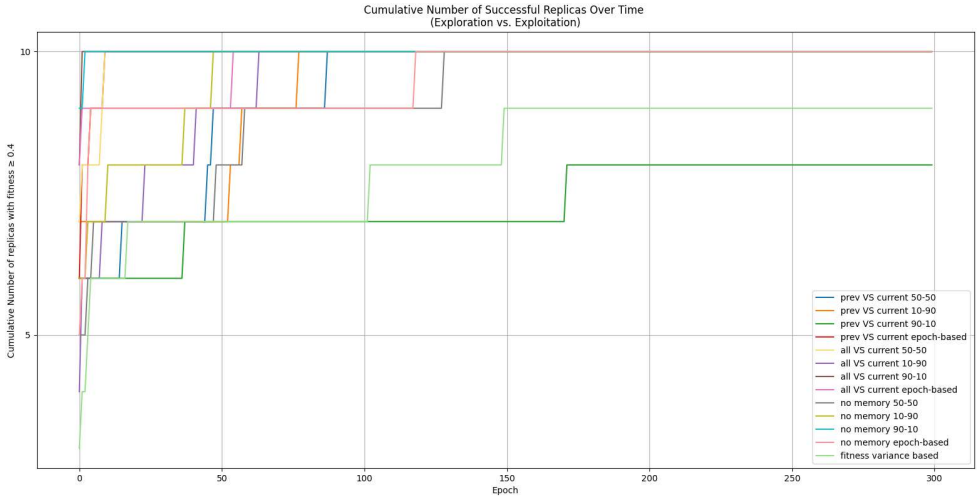

Figure 3.27: Cumulative percentage of successful replicas (RNN Exploration VS Exploitation variants).

During the early epochs, most variants display similar trends, with a steady increase in successful replicas. However, they diverge in long-term performance. The *all vs current 90–10* and *no memory 90–10* variants are the first to reach 100% success. Interestingly, these two strategies represent opposite memory usages; full memory and no memory both benefit from strong exploitation. The *prev vs current 90–10* and *fitness variance-based* strategies lag significantly behind, never achieving full success. While the first one mirrors its poor performance observed in the previous metrics, the second case is more unexpected, having demonstrated consistent behavior in previous metrics. Most remaining strategies all achieve full success with slight differences in convergence timing.

To conclude, across all evaluation metrics used for this analysis, we can observe a few consistent patterns emerging.

First, epoch-based exploration, particularly when combined with memory, consistently led to satisfactory performance.

Second, extreme exploration (as in the 10–90 configurations) proves effective only when supported by memory. The *prev vs current 10–90* variant, for instance,

reaches high peak fitness and exhibits tight distributions. While the same exploration rate without memory, as in *no memory 10–90*, results in lower performance.

Third, extreme exploitation (90–10) yields mixed results. The *all vs current 90–10* is one to achieve full convergence earliest. However, other 90–10 strategies, especially *prev vs current 90–10*, suffer from poor overall performance and inconsistency. Excessive exploitation can be effective either through fortunate early discoveries or when paired with full past memory.

Overall, the most robust and reliable adaptation strategies are those that integrate memory (either of the previous or all past configurations) with a decaying exploration rate over epochs.

The *fitness variance-based* mechanism, while not among the worst performers, fails to match the top ones. Its adaptive logic offers potential. Still, it may require more finely tuned thresholds or dynamic ranges to compete with the best-performing approaches.

3.3.2 Multi-Armed Bandit variants

The following variants presented in this section are inspired by the algorithms presented in the *Multi-Armed Bandits* chapter from the book *Reinforcement Learning: An Introduction* by Richard Sutton and Andrew Barto [SB18]. In Multi-Armed Bandit problems, a learner repeatedly selects from a set of actions (or *arms*) and receives feedback. The reward distribution of each *arm* is unknown, and the goal is to maximize the cumulative reward over time by balancing between exploitation (selecting the best-known arm) and exploration (trying others to gather more information). These methods are particularly suited for this study, where robot controllers must continuously update and select among neural network configurations θ in real time. In this setting, instead of assuming full access to state–action mappings, the problem is reduced to choosing which configuration to reuse (exploitation) or mutate (exploration), similar to selecting among arms that deliver noisy, stochastic rewards.

Three main mechanisms were derived from the book chapter [SB18] and adapted for this setting, the first one being ε-greedy action selection. In this approach, an agent repeatedly selects among discrete actions, or *arms*, by either exploiting the

currently best-known action or exploring alternatives. In our implementation (formally described in Algorithm 17), each action corresponds to a neural network configuration, θ.

The core functioning is similar to previous mechanisms; at each epoch, the current configuration (θ_i) is translated into robot motion behavior and used for a fixed number of steps. The cumulative fitness obtained across these steps is used to compute an average reward (fitness) for that epoch. Then, the value estimate $Q[i]$ for the active configuration θ_i is updated using the standard *sample average* formula [SB18]:

$$Q[i] \leftarrow Q[i] + \frac{r - Q[i]}{N[i]} \tag{3.2}$$

where r is the reward (the observed average fitness for the epoch), and $N[i]$ is the number of times configuration i has been used so far. After updating the Q value for the current configuration, the algorithm decides whether to exploit the best configuration seen so far (i.e., the one with the highest Q) or to explore by generating a new configuration through mutation. The same mutation strategy seen for other RNN variants is applied here as well (see Algorithm 7). This decision is driven by a random value $p \in [0, 1]$ and an exploration probability ϵ. With probability ϵ, exploration is triggered; otherwise, the configuration with the highest current estimate is reused, as dictated by the original ϵ-greedy policy:

- With probability $1 - \varepsilon$, select θ_j where $j = \arg\max_k Q[k]$ (exploitation).
- With probability ε, create $\theta_{i+1} \leftarrow \texttt{mutate}(\theta_i)$ (exploration).

Starting from the baseline algorithm, two sub-variants were implemented, concerning the usage of ϵ. One is the standard ϵ-greedy policy, with a fixed ϵ value, held throughout the run. This was tested with values of $\epsilon \in \{0.1, 0.5, 0.9\}$ to observe the impact of different levels of exploration. The second sub-variant adapts the ϵ value, and consequently the exploration rate, in the $[0.1, 0.9]$ range based on the current epoch:

$$\varepsilon(\texttt{epoch}) = \max(0.1,\ 0.9 - 0.8 \cdot \frac{\texttt{epoch}}{\texttt{MAX\_EPOCHS}}) \tag{3.3}$$

This ensures stronger early exploration, gradually shifting toward heavier exploita-

tion at the end of a run.

The second main variant group takes from the classical ε-greedy selection strategy but replaces the sample-average update rule with an incremental update that uses a constant step-size parameter α, as described in *Reinforcement Learning: An Introduction* [SB18]. This variation is particularly indicated for nonstationary environments, where the reward distribution associated with each arm (in this case, each configuration θ) may change over time.

Instead of computing the mean of all observed rewards for a configuration θ_i, the value estimate $Q[i]$ is updated using the exponentially weighted moving average:

$$Q[i] \leftarrow Q[i] + \alpha \cdot (r - Q[i]) \tag{3.4}$$

where r is the observed reward (average fitness) for the current epoch, and α is a fixed learning rate. This rule assigns greater weight to recent observations, enabling the algorithm to adapt more quickly to changes in performance. In practice, it improves responsiveness at the cost of increased variance.

As in the standard ϵ-greedy formulation, the decision to explore or exploit is governed by the probability ϵ:

- With probability ε, the system explores by mutating the current configuration θ_i to produce a new one.
- With probability $1 - \varepsilon$, it exploits by selecting the configuration θ_j with the highest $Q[j]$.

Even for this mechanism, two different sub-variants were considered, described in Algorithm 18. Similarly to the standard ϵ-greedy variants, in the first one ϵ is fixed for the whole run, and three different exploration rates were tested (0.1, 0.5, and 0.9). As one can imagine, the second sub-variant focuses on adapting the ϵ value: the exploration rate is reduced linearly over time following the same decay schedule defined in Equation (3.3). In both cases, regardless of ϵ, the mechanisms were run and evaluated with two different α values: 0.1 and 0.2.

The third and last Multi-Armed Bandit mechanism explored is the gradient bandit method, which differs essentially from the other two methods seen before. Instead of estimating expected rewards directly for each configuration, this ap-

proach maintains a preference value $H[i]$ for each *arm*, which is updated using the gradient of expected reward relative to $H[i]$.

In this case, action selection is very diverse and is performed probabilistically via a softmax distribution over preferences H:

$$\pi_i = \frac{e^{H[i]}}{\sum_k e^{H[k]}} \tag{3.5}$$

with π_i representing the probability of selecting configuration i at a given epoch. This stochastic policy favors configurations with higher preferences, but still assigns non-zero probability to all others, enabling continual and varied exploration.

After each epoch, preferences are updated to move in the direction of the gradient of the expected reward calculated over H. The update rule is given by:

$$H[i] \leftarrow H[i] + \alpha \cdot (r - B) \cdot (1 - \pi_i) \quad \text{for the selected } i \tag{3.6}$$

$$H[k] \leftarrow H[k] - \alpha \cdot (r - B) \cdot \pi_k \quad \text{for all } k \neq i \tag{3.7}$$

where r is the observed average fitness (reward), B is a running baseline reward (the average of previous rewards), and α is the learning rate controlling the magnitude of preference updates. At each epoch, a configuration is sampled stochastically according to its softmax probability π_i (see Equation (3.5)), which biases selection toward those with higher preferences. The selected configuration is then reused directly if already existing, or mutated into a new configuration if it has not been instantiated before, following the same mutation algorithm as before (see Pseudocode 7). Over time, configurations that yield rewards higher than the baseline are favored, meaning their preference is higher, whereas the preferences for poorly performing configurations are reduced. This implicitly drives the system to concentrate probability mass around effective configurations, while still preserving stochasticity.

This mechanism was run and tested with three values of α: 0.05, 0.1, and 0.2. The whole adaptation mechanism is shown in Algorithm 16.

As for the Exploration vs. Exploitation case (Section 3.3.1), each of these variants was tested and evaluated on 10 independent runs, using the same arena (see Figure 2.4), with a fixed hidden layer size of $H = 10$ for the NNs, and uniform

Algorithm 16: Online adaptation based on gradient bandit

```
H ← array of preferences initialized to 0
B ← 0 // running baseline average reward
θ_i ← randomly initialized configuration
for epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for step in 1 to MOVE_STEPS do
        sense environment → (s_L, s_R)
        (outL, outR) ← RNN_forward
        set motor speeds using outputs
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum/MOVE_STEPS
    B ← B + (avg_fitness − B)/(epoch+1)
    compute softmax probabilities π_i ← e^{H[i]} / Σ_k e^{H[k]}
    foreach configuration k do
        if k = i then
            H[k] ← H[k] + α · (avg_fitness − B) · (1 − π_k)
        else
            H[k] ← H[k] − α · (avg_fitness − B) · π_k
        end
    end
    sample configuration j according to π_k
    θ_{i+1} ← mutated copy of θ_j (if new)
end
```

additive noise (± 0.01) for proximity sensors and wheel actuators.

Final Fitness Sum Distribution Figure 3.29 presents the distribution of the final sum of fitness for all the variants and cases tested.

Variants based on standard ϵ-greedy selection generally achieved strong performance, with $\epsilon = 0.1$ obtaining one of the highest median sums and tightest IQR, indicating both excellent efficiency and stability. Larger ϵ values (0.5, 0.9) show slightly more variability and lower medians, indicating that performance tends to decline as exploration rate increases. Still, they all reach satisfying performances.

For ϵ-greedy with α, performance depends heavily on the learning rate. The

combination $\alpha = 0.1$, $\epsilon = 0.1$ is the worst performer overall, with a low median, very low outliers, and a significantly wider spread compared to others. Increasing α to 0.2 slightly improves the median but also introduces substantial variance, both with $\epsilon = 0.1$ and $\epsilon = 0.9$. Using an epoch-adaptive ϵ significantly helps both $\alpha = 0.1$ and $\alpha = 0.2$ cases, with the latter outperforming all the others in terms of median.

Gradient bandit methods show moderate to good total fitness depending on the learning rate. While $\alpha = 0.05$ and $\alpha = 0.1$ have lower medians, $\alpha = 0.2$ reaches honorable total sums with good consistency.

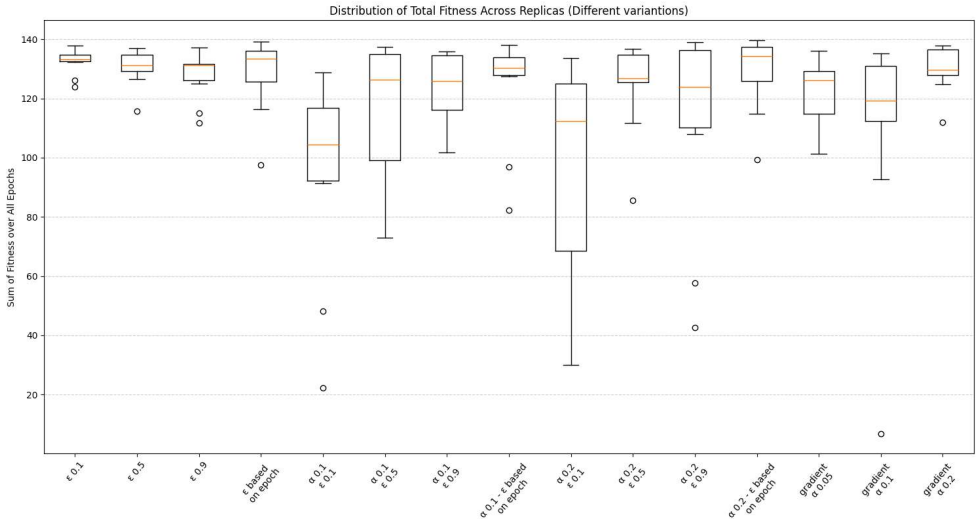

Figure 3.28: Distribution of total fitness across replicas (Multi-Armed Bandit variants).

Maximum Fitness Distribution The distribution of the maximum fitness values obtained from the Multi-Armed Bandit variants is reported in Figure 3.22.

Standard ϵ-greedy selection variants again show great overall performance with very high medians, high top and bottom whiskers, and minimal box plot spreads. The decaying ϵ case median slightly outperforms the others, obtaining the highest maximum fitness median value of all.

The α variants struggle more with maximum fitness, especially when paired with an ϵ value of 0.1, bringing the lowest maxima and widest spread results.

Higher values of ϵ or adaptive ϵ strategies clearly improve performance, with $\alpha = 0.2$ adaptive ϵ reaching the highest values in this group, exhibiting particularly good stability as well.

Gradient bandit strategies showcase average performance overall, with $\alpha = 0.2$ performing best. Lower α values tend to exhibit some inconsistent outcomes, possibly due to slower convergence.

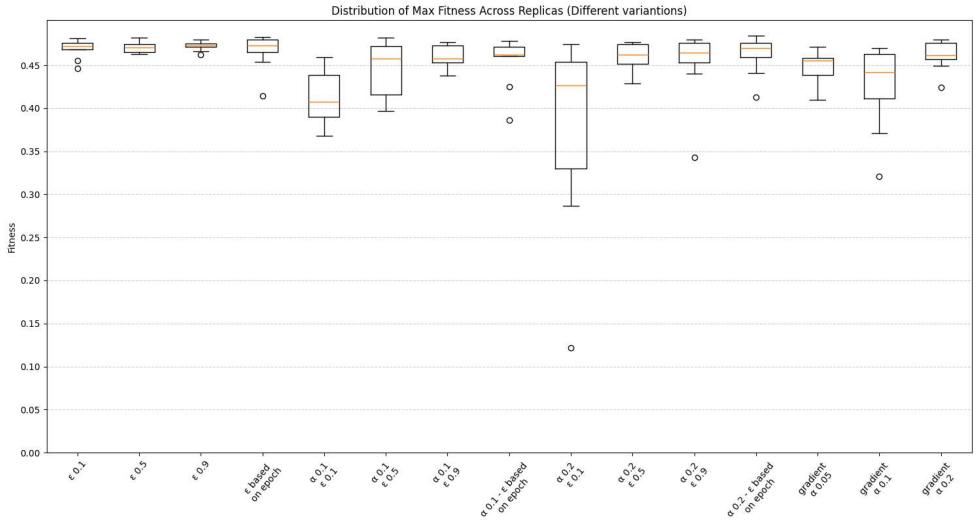

Figure 3.29: Distribution of maximum fitness across replicas (Multi-Armed Bandit variants).

Cumulative % of Successful Replicas Figure 3.30 shows the evolution of the cumulative percentage of successful replicas, defined as those reaching average fitness ≥ 0.4 over training epochs.

All standard ϵ-greedy variants reach 100% success, with $\epsilon = 0.1$ being the fastest. $\epsilon = 0.5$ and $\epsilon = 0.9$ reach complete success around epoch 70, while the decaying-ϵ strategy was slower, but still converged fully in the first half of the total epochs.

For the α variants, only a few managed full convergence: ($\alpha = 0.1$, $\epsilon = 0.9$), ($\alpha = 0.2$, $\epsilon = 0.5$), and ($\alpha = 0.2$, ϵ epoch-based). The worst case ($\alpha = 0.1$, $\epsilon = 0.1$) only reached 50% success.

Gradient bandit strategies also yielded varied outcomes. The $\alpha = 0.05$ and $\alpha = 0.2$ cases reached full success very early, in the first few epochs, while $\alpha = 0.1$

stagnated at around 75%, possibly due to some sensitivity to the choice of learning rate.

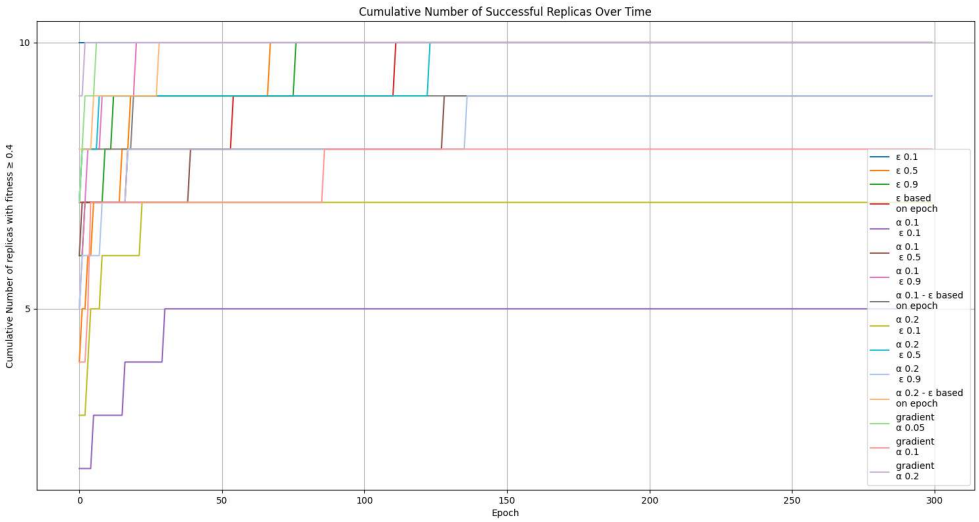

Figure 3.30: Cumulative percentage of successful replicas (Multi-Armed Bandit variants).

To conclude this analysis, among the Multi-Armed Bandit variants explored, the most effective mechanisms, particularly in terms of the final fitness sum, were those that incorporated a decaying exploration probability over time. The top three performers were $\alpha = 0.2 + \epsilon$ decaying with epochs, ϵ decaying with epochs, and $\epsilon = 0.1$, which consistently achieved high median values across all evaluation metrics. In contrast, the worst-performing group was the ϵ-greedy with constant α family. These methods are designed for nonstationary problems, but in this case, the environment and the reward function (fitness) don't change drastically across epochs. The one interesting exception to the group is the $\alpha = 0.2 +$ *decaying* ϵ variant, which benefits from high early exploration to diversify its configuration pool and then reduces randomness in later epochs, stabilizing the effect of large α updates. Thus, despite the Q-values being updated with a high α, having fewer new random configurations limits instability. Similarly, $\alpha = 0.1 +$ decaying ϵ outperformed its static-ϵ siblings but without ever overthrowing the top performers.

Algorithm 17: Online adaptation - ϵ-greedy

```
Q, N ← arrays for value estimates and counts
θ_i ← randomly initialized configuration
Q[i] ← 0, N[i] ← 0
for epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for step in 1 to MOVE_STEPS do
        sense environment → (s_L, s_R)
        (outL, outR) ← RNN_forward(s_L, s_R, θ_i)
        set motor speeds using outputs
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum/MOVE_STEPS
    N[i] ← N[i] + 1 ,  Q[i] ← Q[i] + (avg_fitness − Q[i]) / N[i]
    p ← random uniform in [0, 1]
    if adaptive ε then
        ε ← max(0.1, 0.9 − 0.8 · epoch / MAX_EPOCHS)
        if p ≤ ε then
            θ_{i+1} ← mutate(θ_i) ,  Q[i+1] ← 0, N[i+1] ← 0
        end
        else
            θ_i ← θ_j where j = arg max_k Q[k]
        end
    end
    else
        if (ε = 0.1 and p < 0.1) or (ε = 0.5 and p ≤ 0.5) or (ε = 0.9 and
         p ≤ 0.9) then
            // explore
        else
            // exploit
        end
    end
end
```

Algorithm 18: Online adaptation - ϵ-greedy with α Q-value updates

```
Q ← array of estimated values for each configuration
θ_i ← randomly initialized configuration
Q[i] ← 0
for epoch in 1 to MAX_EPOCHS do
    fitness_accum ← 0
    for step in 1 to MOVE_STEPS do
        sense environment → (s_L, s_R)
        (outL, outR) ← RNN_forward
        set motor speeds using outputs
        fitness_accum ← fitness_accum + evaluate_performance()
    end
    avg_fitness ← fitness_accum/MOVE_STEPS
    Q[i] ← Q[i] + α · (avg_fitness − Q[i])
    p ← random uniform in [0, 1]
    if adaptive ε then
        ε ← max(0.1, 0.9 − 0.8 · epoch/MAX_EPOCHS)
        if p ≤ ε then
            θ_{i+1} ← mutate(θ_i) ,  Q[i+1] ← 0
        end
        else
            θ_i ← θ_j where j = arg max_k Q[k]
        end
    end
    else
        if (ε = 0.1 and p < 0.1) or (ε = 0.5 and p ≤ 0.5) or (ε = 0.9 and
          p ≤ 0.9) then
            // explore
        else
            // exploit
        end
    end
end
```

3.4 Additional Experiments with Best Five

From all the previous RNN variants, we selected five based on the obtained results. We used them to conduct further experiments, aiming to gain insights into their robustness and adaptivity when multiple robots are involved or malfunctions may occur.

The selected five are:

- the exploration vs. exploitation variant with -1 memory and epoch-based exploration rate: previous VS current + exploration % based on epoch,
- the exploration vs. exploitation variant with *full* memory and epoch-based exploration rate: all VS current + exploration % based on epoch,
- the multi-armed bandit variant with α value at 0.2 and ϵ value based on epochs,
- the multi-armed bandit variant with ϵ value based on epochs,
- the multi-armed bandit variant with a fixed ϵ of 0.1.

3.4.1 Multiple Robots

The first type of experiment involves having multiple robots in the arena, either using their own mechanisms or acting as wandering, moving obstacles. The primary goal is to assess how each mechanism performs in varied multi-agent settings where other robots act as moving obstacles, potentially altering the difficulty and dynamics of the obstacle avoidance task.

5 robots - 5 mechanisms

Five robots are placed simultaneously in the same arena (see Figure 2.4) and each is assigned one of the five selected adaptive mechanisms listed earlier. At the beginning of each run, robots are randomly positioned and oriented within the environment, and no external synchronization or communication is used among them.

Each robot operates independently, maintaining its own online adaptation loop, fitness computation, and internal data structures. Controllers evolve their behavior through the same adaptation logic, configuration, and architecture used in isolation.

This experiment was conducted through 10 independent runs, meaning that for each variant, we collected data from 10 replicas where all mechanisms were operating simultaneously. Each robot maintained separate data structures and logging to collect and analyze data independently per mechanism. All mechanisms had a fixed hidden layer size of $H = 10$, and uniform additive noise (± 0.01) for proximity sensors and wheel actuators.

By placing heterogeneous adaptive strategies in direct interaction, we aim to evaluate their robustness, adaptability, and tolerance under shared environmental constraints using a set of metrics equivalent to those previously employed.

Final Fitness Sum Distribution The distribution of final cumulative fitness for each robot in the shared environment is depicted in Figure 3.31.

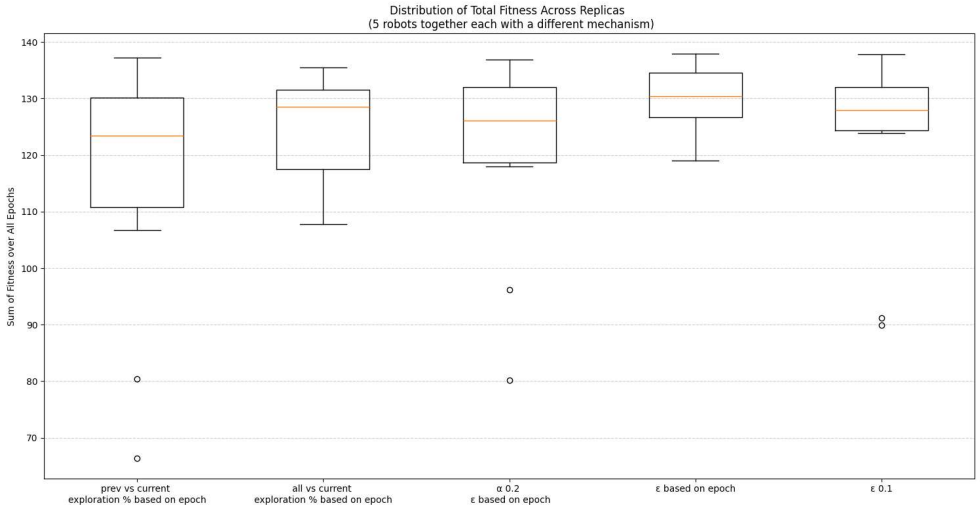

Figure 3.31: Distribution of total fitness across replicas (5 robots - 5 controllers scenario).

All five mechanisms achieve relatively high total fitness scores, despite the coexisting condition. Among them, the best performer is the *decaying ϵ-greedy*,

which produces the highest median total fitness along with a tight IQR, indicating great performance and consistency across replicas. This is followed by the *all vs. current + epoch-based exploration rate* mechanism, which has the second-highest median but also a considerably larger IQR, pointing to less stability. The $\epsilon = 0.1$-*greedy* variant's performance is satisfying, having the smallest box plot spread and the third-highest median. The worst one in this case is *previous vs. current + epoch-based*, presenting the lowest median and the widest IQR, suggesting poorer performance and less stability across runs.

Maximum Fitness Distribution Distribution of the maximum fitness values is shown in Figure 3.32.

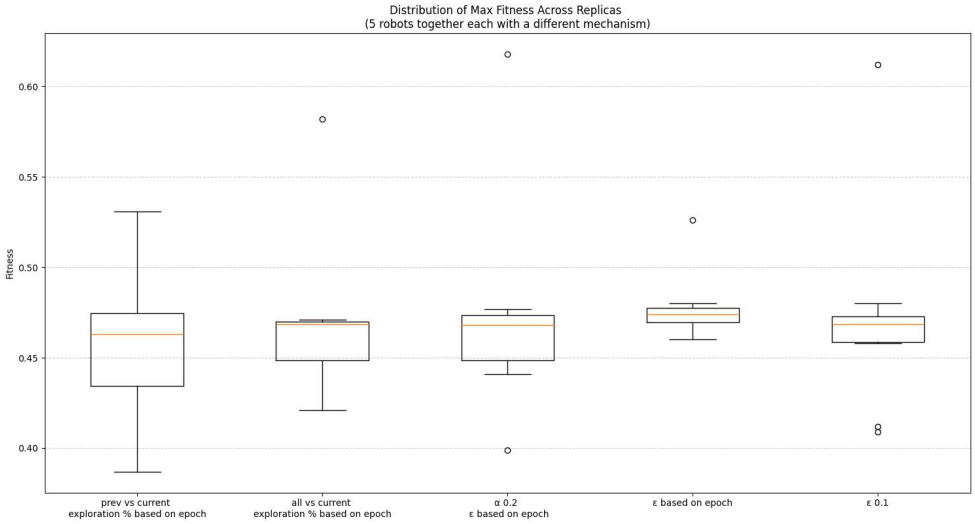

Figure 3.32: Distribution of maximum fitness across replicas (5 robots - 5 controllers scenario).

All mechanisms produced peak fitness values within a similar range (around 0.45–0.5), with occasional outliers exceeding this range up to ≈ 0.6. The *previous vs. current + decaying exploration rate* variant exhibits a median value close to the others, but simultaneously displays the highest and lowest fitness values, indicating high variance. By contrast, the *decaying* ϵ*-greedy* and $\epsilon = 0.1$-*greedy* strategies delivered the most stable performance, combining high median values with compact IQR.

Cumulative % of Successful Replicas Figure 3.30 shows the evolution of the cumulative percentage of successful replicas, defined as those reaching average fitness ≥ 0.4 over epochs.

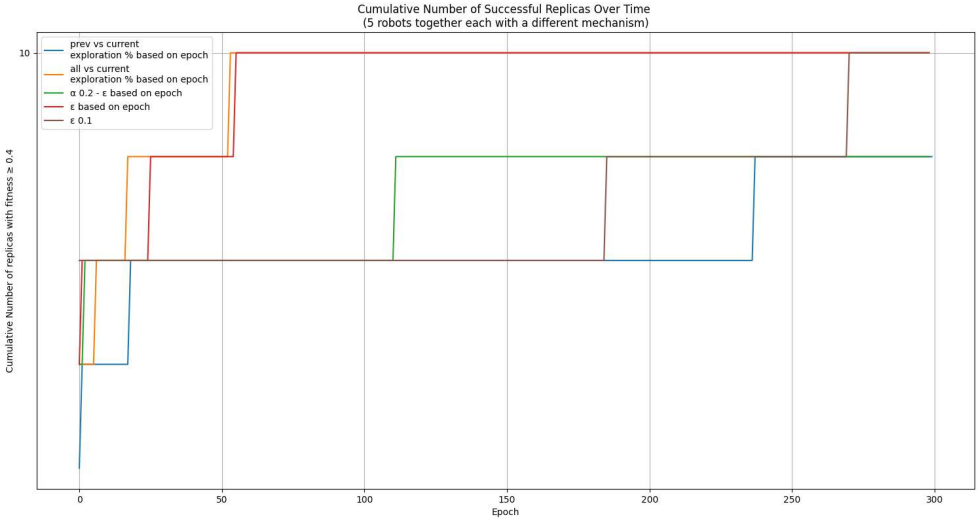

Figure 3.33: Cumulative percentage of successful replicas (5 robots - 5 controllers scenario).

In this case, *all vs. current + epoch-based* and *decaying* ϵ*-greedy* were the earliest to reach 100% success, almost at the same time. Despite $\epsilon = 0.1$-*greedy* starting with the highest initial success rate, it only completed its convergence at the end, after epoch 250. To conclude, $\alpha = 0.2 +$ *decaying* ϵ and *previous vs. current + epoch-based exploration* failed to reach full success within the given time frame, stabilizing below the other, both at $\approx 75\%$.

1 mechanism - wandering obstacles

In this second experiment, a single robot is placed in the arena (Figure 2.4) and controlled by one of the five selected mechanisms. To simulate a dynamic and unpredictable environment, four additional robots of the same kind are placed as moving obstacles; they do not perform any learning or evaluation but instead act as distractors. At each step, one of their two wheels is randomly chosen and assigned a new velocity uniformly sampled from the $[0, 10]$ range, simulating a random walk.

For each of the five mechanisms, we executed 10 independent runs, exploiting a fixed hidden layer size of $H = 10$, and uniform additive noise (± 0.01) for proximity sensors and wheel actuators. The comparative analysis is based on the same performance metrics adopted for the previous experiment.

Final Fitness Sum Distribution Figure 3.34 illustrates the distribution of final cumulative fitness for each mechanism when encountering moving obstacles.

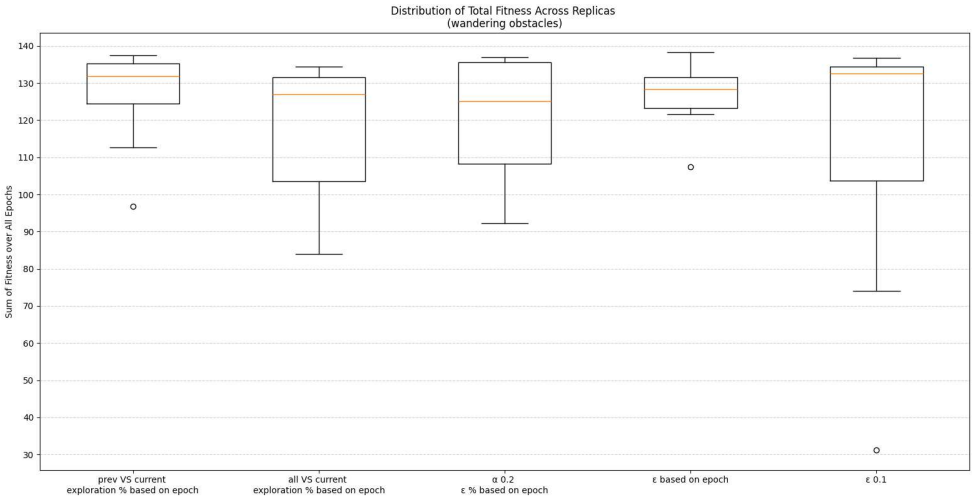

Figure 3.34: Distribution of total fitness across replicas (1 controller with wandering obstacles scenarios).

All mechanisms exhibit relatively high total fitness scores in this setting, and their medians are quite similar to each other. The fixed $\epsilon = 0.1$-*greedy* variant shows the highest median total fitness but the widest spread as well, characterized by a wide IQR, low bottom-whisker, and the presence of an extremely low outlier. The *previous vs current + decaying exploration %* mechanism ranks second in terms of median and shows more consistent performance with a narrower IQR and box plot spread. The *decaying ϵ-greedy* variant presents the third-best median, as well as achieving very contained IQR and whiskers, resulting in the smallest spread. Oppositely, *all vs current + decaying exploration %* and the $\alpha = 0.2$ + *epoch-based* ϵ yielded the lowest medians and broad IQRs.

Maximum Fitness Distribution Figure 3.34 shows the distribution of the maximum fitness values obtained by each mechanism during the runs.

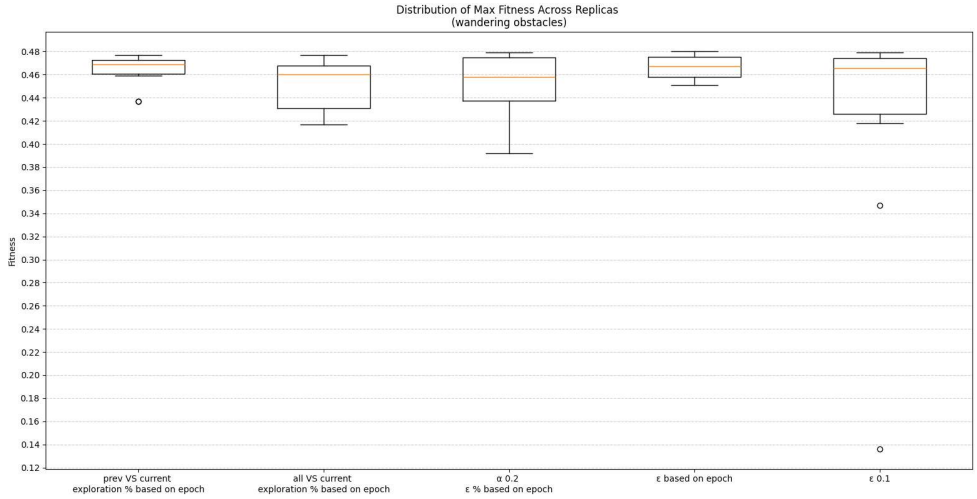

Figure 3.35: Distribution of maximum fitness across replicas (1 controller with wandering obstacles scenarios).

The highest maximum fitness values are observed in *previous vs current + epoch-based exploration* and *decaying ϵ-greedy*, both of which also report the tightest IQRs and whiskers; the former slightly outperforms the latter in terms of median. All the other mechanisms exhibit satisfactory performance, showing more variation, with their medians being below the two best but still close to them. Among these, $\epsilon = 0.1$**-greedy** displays the longest IQR and very low outliers.

Cumulative % of Successful Replicas The final metric focuses on the cumulative percentage of successful replicas, shown in Figure 3.36.

The fastest convergence to 100% of successful replicas (fitness ≥ 0.4) is achieved by both *previous vs current + epoch-based exploration* and *decaying ϵ-greedy* almost simultaneously and very early, with all replicas reaching the threshold within the first 25 epochs. In contrast, both $\alpha = 0.2+$ *epoch-based* ϵ and $\varepsilon = 0.1$-*greedy* never reach full success rate during the simulation, even though the $\alpha = 0.2$ variant seems to be performing better than *decaying ϵ-greedy*, in the first few epochs. The *all vs current* variant lags in the early epochs but manages to reach 100% in a short amount of time, around epoch 80.

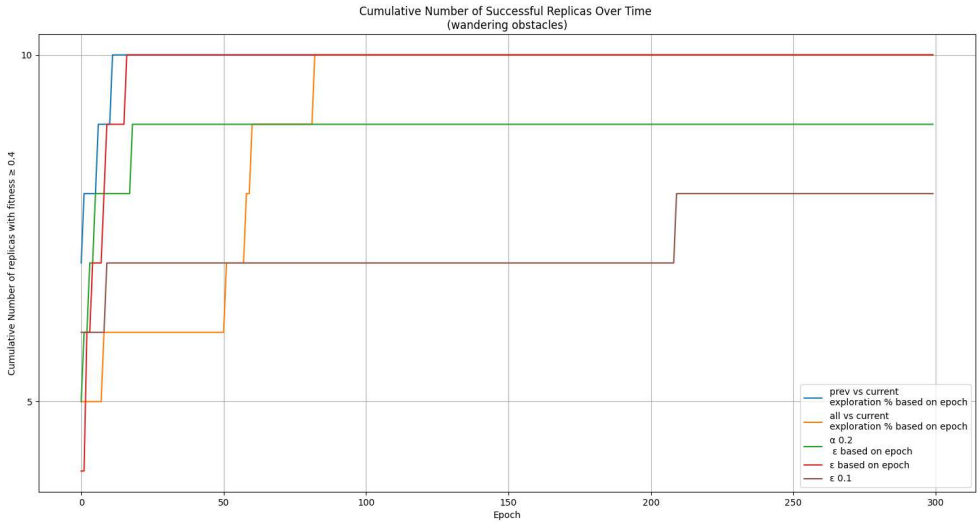

Figure 3.36: Cumulative percentage of successful replicas (1 controller with wandering obstacles scenarios).

1 mechanism with best configurations initialization - wandering obstacles

As a final experiment for this section, we want to check how well each adaptation mechanism can maintain performance when starting from a previously learned optimal configuration. For each mechanism, 10 replicas of the single-robot static environment case were analyzed to extract the highest-performing configuration from each. These top configurations were then reused as initialization points for new simulations under dynamic conditions, where one of the five best mechanisms is assigned to a robot, and the other four robots act as wandering, moving obstacles.

For each of the five mechanisms, we executed 10 independent runs (so that each of the 10 best configurations found can be exploited and observed exactly once), setting a fixed hidden layer size of $H = 10$, and uniform additive noise (± 0.01) for proximity sensors and wheel actuators. The comparative analysis is based on the same performance metrics adopted for the previous experiment.

Final Fitness Sum Distribution Figure 3.37 illustrates the distribution of final cumulative fitness for each mechanism when dealing with moving obstacles and initialized with a previously identified optimal configuration.

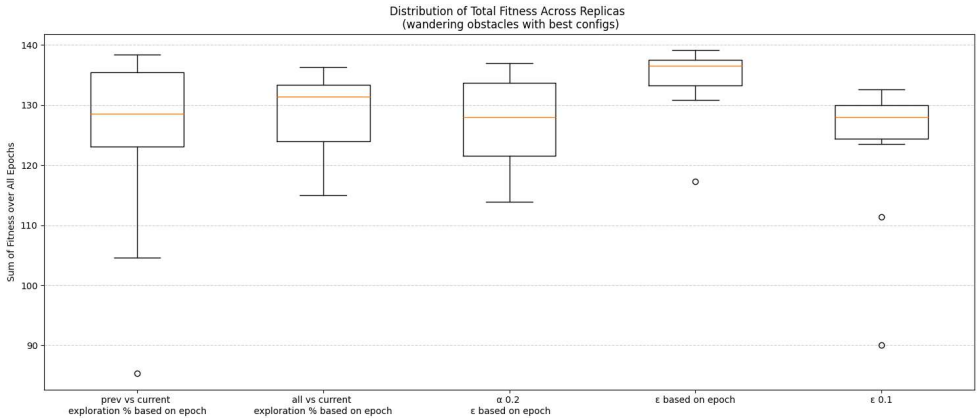

Figure 3.37: Distribution of total fitness across replicas (1 controller starting with best configurations with wandering obstacles scenario).

All mechanisms achieve a strong overall performance. Most notably, compared to the random initialization case, all mechanisms except for *previous vs current* have reduced box plot spreads. The *all vs current*, $\alpha = 0.2$ + ϵ *based on epoch*, and *decaying* ϵ*-greedy* mechanisms also show improved median total fitness values. Whereas, $\epsilon = 0.1$-*greedy* displays a slightly lower median but benefits from greatly reduced variance. Its IQR is tighter, and although outliers are still present, they are substantially less extreme than in the random initialization scenario. The best-performing method by median is *decaying* ϵ*-greedy*, while the worst total fitness medians are brought by *previous vs current* and $\alpha = 0.2$ + *decaying* ϵ.

Maximum Fitness Distribution Figure 3.38 shows the distribution of the maximum fitness values obtained by each mechanism.

The highest maximum fitness median and the smallest spread are again achieved by *decaying* ϵ*-greedy*, confirming its stability and high performance. The worst results are obtained by $\epsilon = 0.1$-*greedy*, with the worst median and some low outliers, and *previous vs current*, which has the second-lowest median and the largest IQR and whiskers. All other mechanisms — *all vs current*, $\alpha = 0.2$ + *decaying* ϵ, and *decaying* ϵ*-greedy* show improved maximum fitness medians compared to the random initialization case, presented earlier.

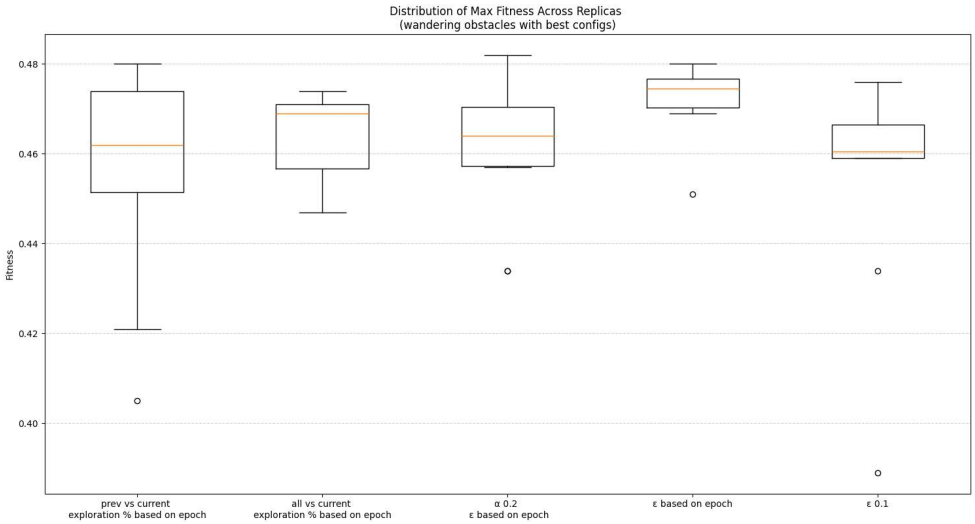

Figure 3.38: Distribution of maximum fitness across replicas (1 controller starting with best configurations with wandering obstacles scenario).

Cumulative % of Successful Replicas The final metric focuses on the cumulative percentage of successful replicas, and is shown in Figure 3.39.

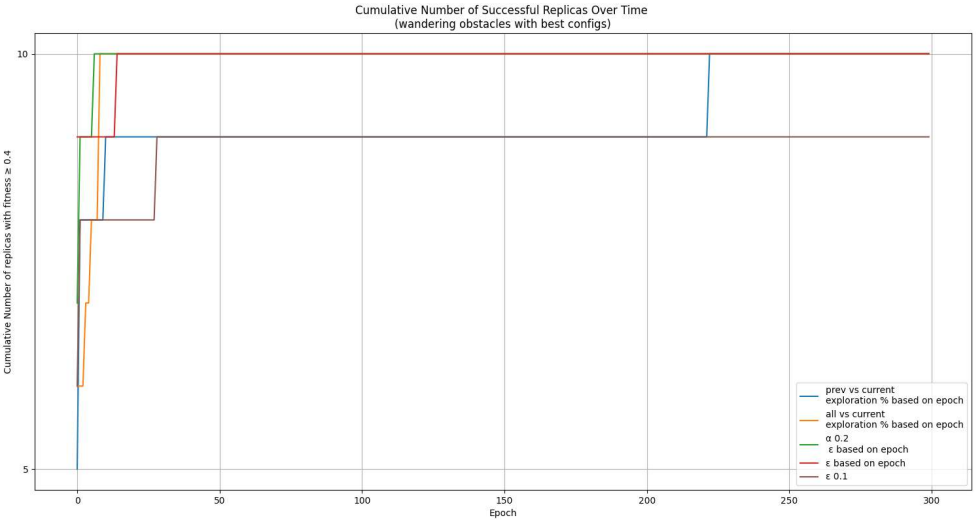

Figure 3.39: Cumulative percentage of successful replicas (1 controller starting with best configurations with wandering obstacles scenario).

Here, $\epsilon = 0.1$-*greedy* never reaches 100% success, although it performs strongly during early epochs. Other mechanisms like $\alpha = 0.2$ + *decaying* ϵ, *all vs current*,

and *decaying ϵ-greedy* all reach full success notably early, within the first 15 epochs. This represents a significant improvement over their performance in the previous test. On the contrary, *previous vs current* mechanism performs worse in this experiment, achieving full success much later (around epoch 200–250), whereas it reached 100% very early in the random initialization setting.

To conclude this analysis, among the five selected mechanisms, *decaying ϵ-greedy* is the overall best performer. It maintained high median values with tight spreads across all scenarios, and it was the only method to reach 100% successful replicas early and consistently, regardless of the environment or initialization strategy. The *all vs. current* variant also showed strong performance, although its variance was occasionally higher than that of ϵ-based methods; it improved when initialized with good configurations. The *constant-α ($\alpha = 0.2$) with decaying ϵ* strategy yielded mixed results: while it had moderate medians and convergence speed when randomly initialized, it clearly benefited from initialization with good configurations. The *previous vs. current* strategy showcased the highest instability during multi-controller tests, with the lowest medians and largest variance. However, it proved very competitive in the wandering-obstacle scenario with random initialization. For this mechanism, starting from its best configurations did not yield further improvements but rather deteriorated performance. Finally, the fixed $\epsilon = 0.1$-*greedy* mechanism is considered the weakest performer: it either failed to reach full convergence or did so only in later epochs, and it repeatedly produced low outliers. Although it occasionally showed high median values, these good results were always paired with high instability as well. Its performance improved slightly when initialized with its optimal configurations, but it still lagged behind the top-performing strategies.

3.4.2 Wheels and Sensors Malfunctions

To further test the robustness and adaptability of the five selected mechanisms, we simulate two types of malfunctions in the system through sensor noise and actuator noise. These disturbances were designed to simulate realistic hardware imperfections and assess each controller's ability to cope with degraded input or execution.

To simulate proximity sensor malfunctioning, random uniform noise in the range $[-0.1,\ 0.1]$ was injected into the virtual proximity sensor readings. While for the wheel actuator faults, uniform noise in the same $[-0.1,\ 0.1]$ range was added to the computed wheel velocities.

The different types of malfunction were kept separate, so for each kind and each mechanism, 10 independent runs were executed. For both cases, in each replica, only a single side (left or right) was affected to assess asymmetrical disturbances. Specifically, for sensor noise, the disturbance was applied to the left-side sensors in 5 replicas and to the right-side sensors in the remaining 5. In this scenario, the NNs operate with altered proximity sensor values. Similarly, for actuator noise, 5 runs had noise added to the left wheel and 5 to the right wheel. The perturbed wheel velocities were used in both the neural control phase (for obstacle avoidance) and the random exploration phase (when no obstacles were detected).

Final Fitness Sum Distribution Figure 3.40 presents the total fitness distribution in the presence of selective virtual proximity sensor noise.

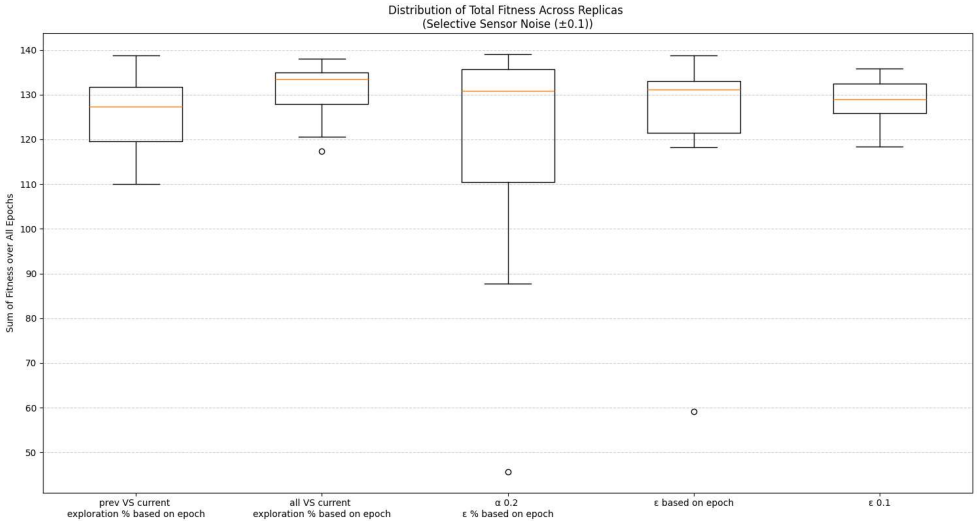

Figure 3.40: Distribution of total fitness across replicas (sensor malfunctions scenarios).

Among the tested mechanisms, *all VS current with epoch-based exploration %* and *decaying ϵ-greedy* appear as the most robust, achieving the highest median values and maintaining tight IQRs. They both present outliers; *decaying ϵ-greedy*'s

outlier is very close to the minimum, while for *all VS current* the outlier is considerably lower. The $\epsilon = 0.1$-*greedy* variant performs comparably; its box plot spread is similar to *all VS current*, it presents no outliers, but the median is slightly lower than the best. In contrast, the *previous VS current* mechanism yields the lowest median fitness, while $\alpha = 0.2$ + *decaying* ϵ shows high variability; its median is relatively strong, but it suffers from the widest spread and the most extreme outlier, indicating a highly unstable behavior.

In this case, the best-performing mechanism in terms of median is $\alpha = 0.2$ + *decaying* ϵ, reaching the highest median value of all mechanisms. However, its performance is damaged by a significantly broad inter-quartile range and a severe low outlier. *All VS current*'s median follows closely after, showing a shorter IQR as well. Even in this case, the $\epsilon = 0.1$-*greedy* variant delivers the weakest results, with the lowest median and the largest spread. Both *decaying* ϵ-*greedy* and *prev VS current* show average results. Their median fitness values aren't low, but are still below the top-performing strategies.

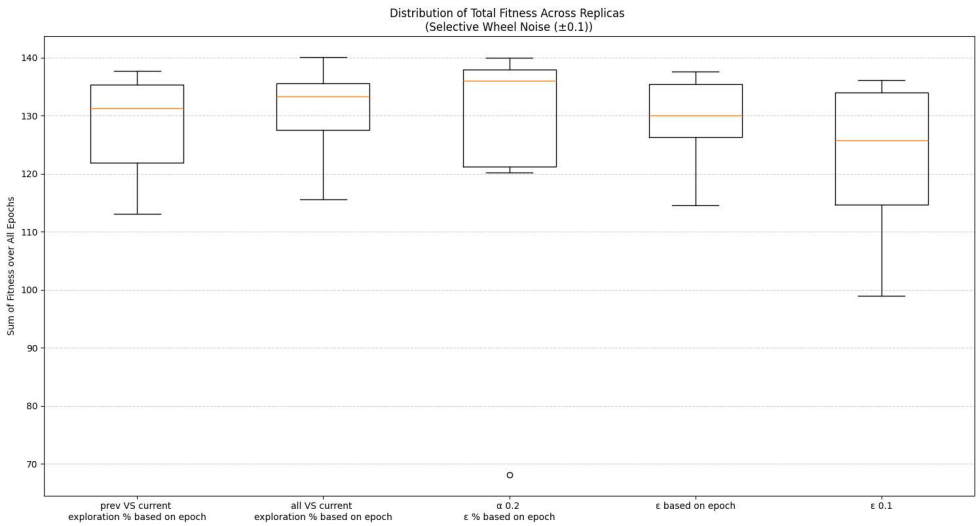

Figure 3.41: Distribution of total fitness across replicas (wheel malfunctions scenarios).

Maximum Fitness Distribution Figure 3.42 and Figure 3.43 display the distribution of the highest fitness values obtained during each run, under sensor and actuator noise, respectively.

Under sensor noise, all mechanisms demonstrate strong performance, all reaching maximum values in the 0.47–0.48 range during their executions. The best-performing methods in terms of median values are *all vs. current* and $\alpha = 0.2$ *with decaying* ϵ, both of which achieve the highest medians. However, the latter has a slightly lower median, exhibits a wider IQR, and has a notably low-end outlier, suggesting less stable performance. These are closely followed by *prev vs. current* and *decaying* ϵ*-greedy*. Their medians are lower but still considerably high. Moreover, their overall box plot spreads are relatively tight. The last one is the $\epsilon = 0.1$-*greedy* variant, which records the lowest median. Although the difference with the others is marginal, it still maintains high values across runs and shows good stability.

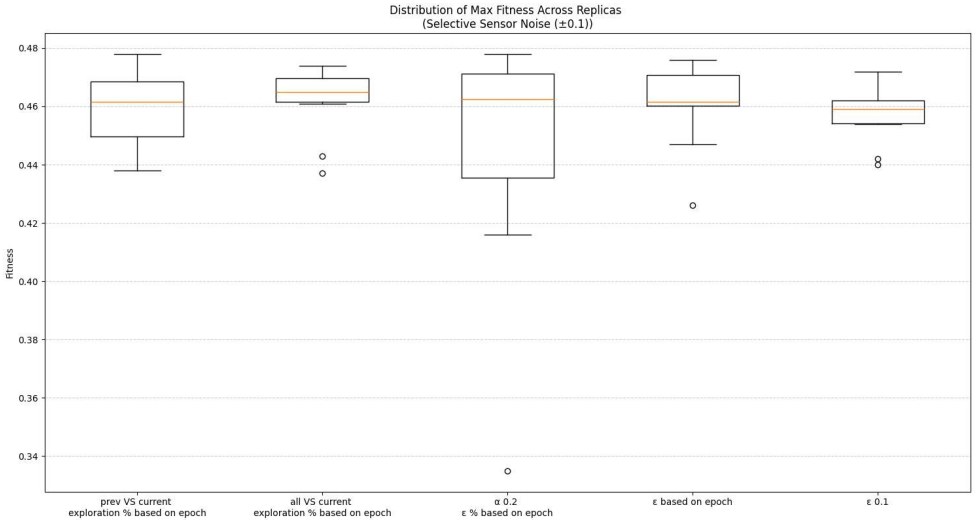

Figure 3.42: Distribution of maximum fitness across replicas (sensor malfunctions scenarios).

When dealing with wheel noise, once again, $\alpha = 0.2$ *with decaying* ϵ achieves the highest median and some of the highest individual fitness values. However, its broader spread and low outlier mark it as less reliable. On the contrary, *all vs. current* and *decaying* ϵ*-greedy* show more consistency. Their median values are immediately below the highest, but their IQRs and whiskers are more compact. The $\epsilon = 0.1$ mechanism, on the other hand, yields the lowest median and widest variability.

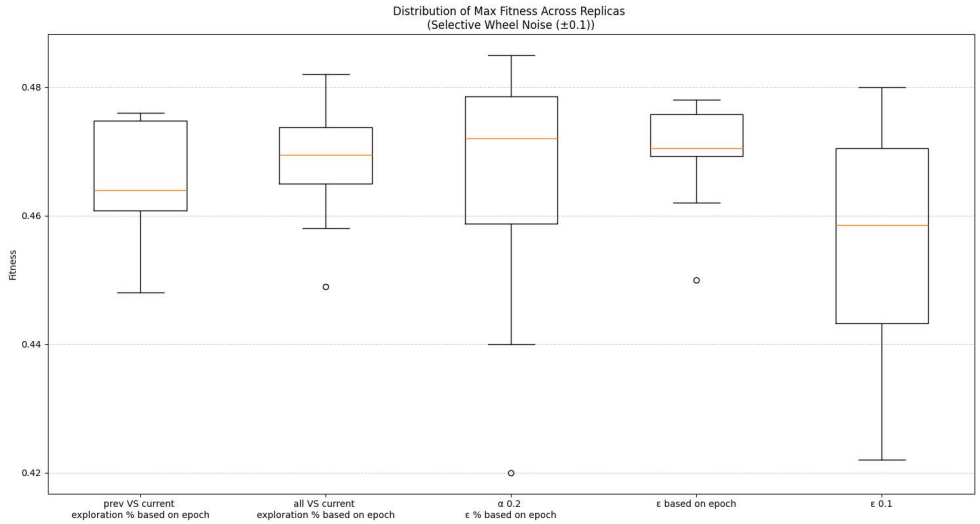

Figure 3.43: Distribution of maximum fitness across replicas (wheel malfunctions scenarios).

Cumulative % of Successful Replicas The final metric focuses on the cumulative percentage of successful replicas, shown in Figure 3.44 and Figure 3.45.

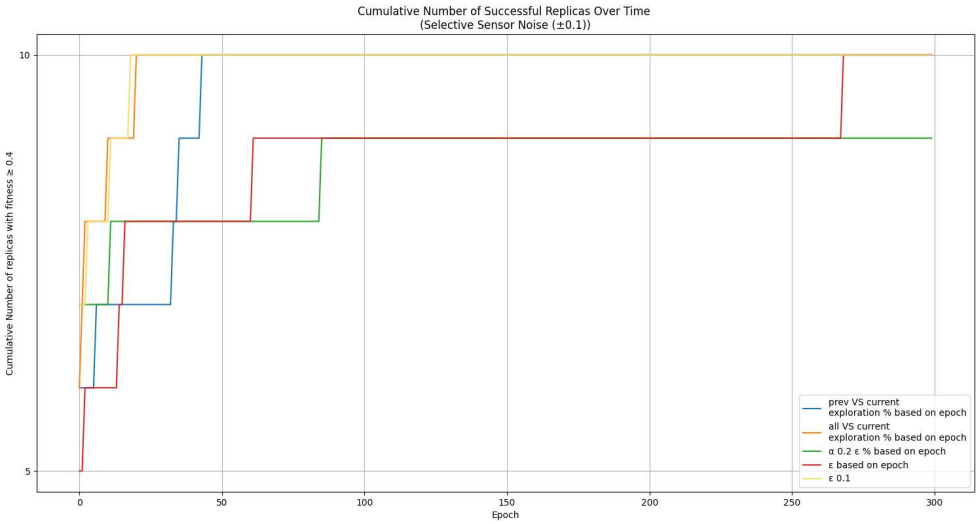

Figure 3.44: Cumulative percentage of successful replicas (sensor malfunctions scenarios).

When selective sensor noise is applied, $\epsilon = 0.1$ and *all VS current* show the

fastest convergence, achieving full success within the first 30 epochs. *Prev VS current* follows closely, reaching 100% before one-sixth of epochs, despite having lower median fitness in other metrics. In contrast, *decaying ϵ-greedy* and $\alpha = 0.2$ + *decaying* ϵ have a strong early start but stall below full success for most of the run. While the former ultimately reaches 100% in the last epochs, the latter plateaus at 90% and never converges fully

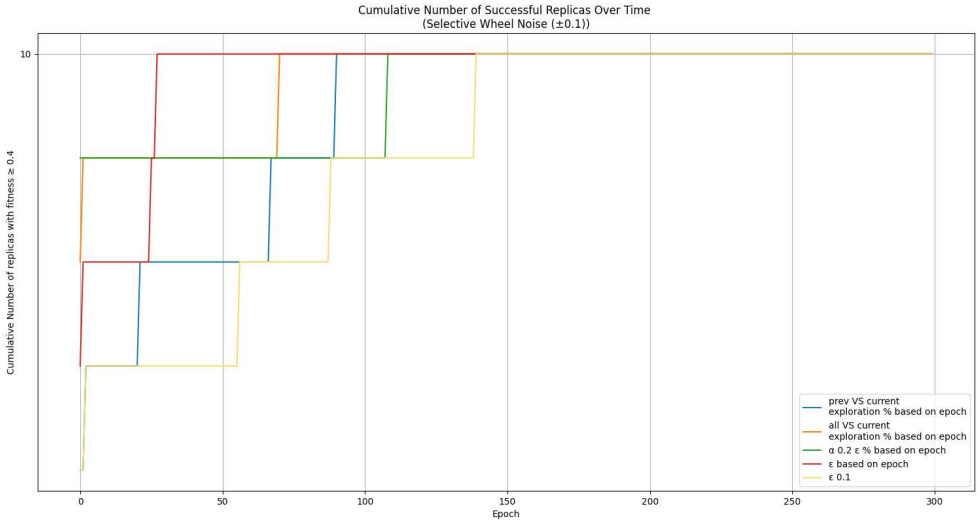

Figure 3.45: Cumulative percentage of successful replicas (wheel malfunctions scenarios).

Interestingly, under selective noise for wheel actuators, all mechanisms eventually reach 100% success. Notably, *decaying ϵ-greedy* starts relatively lower but is the first to achieve full success. $\alpha = 0.2$ + *decaying* ϵ and *all VS current* initially perform better but converge later. The last one to reach full convergence is $\epsilon = 0.1$; it starts low and always trails behind the others, but it eventually reaches 100% of successful replicas.

To conclude this last analysis, we can state that across both malfunction settings, *all vs current with decaying exploration rate* showed the best performance, consistently combining high median fitness, low variance, and fast convergence. This stable performance under both sensor and actuator disturbances marks it as the most robust and effective strategy in these experiments. The $\alpha = 0.2$ variant also achieved high medians but at the cost of greater variability and slower conver-

gence, making it less reliable despite its potential. *Decaying ϵ-greedy* performed well overall, showing both strong fitness and stability, although it never ranked first. *Prev vs current* performance delivered average performance across all metrics, never excelling nor failing completely. Finally, $\epsilon = 0.1$ can be considered the worst one, often presenting weaker performance and greater instability, especially under wheel noise.

Chapter 4

Thymio Demo

To check and validate the feasibility of the proposed online adaptation mechanisms in a real-world scenario, we conducted a live demonstration using a physical *Thymio* robot controlled by the *decaying ϵ-greedy* mechanism. The experiment replicated the simulation environment used with *Footbot* robots, by recreating the original simulation arena (see Figure 2.4) as a similar $1m \times 1m$ physical arena with similar obstacles.

The chosen mechanism was adapted to the *Thymio* robot while maintaining the same logic flow explained in Algorithm 17. The main adaptations regard the proximity sensors used and the range of possible velocity values for the robot wheels. While for the *Footbot* we exclusively used positive velocities only, in the $[0, 10]$ range; for the *Thymio* to move correctly and extensively, we need both positive and negative velocity values. So, the range of possible wheel velocities was changed to a $[-300, 300]$ range. As a consequence, the fitness evaluation function was restored to its original form to account for the negative values as well:

$$\Phi = V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - i) \tag{4.1}$$

To qualitatively illustrate the evolution of robot behavior over time, we recorded the robot's motion at three points during execution: at epoch 0, epoch 50, and epoch 100. These recordings are available at Epoch 0, Epoch 50, and Epoch 100 respectively. As can be seen from the first video, the robot starts with a random configuration biased towards forward movement and left turns. At epoch 0, this

bias causes the robot to collide with an obstacle on its left side, and it will take some mutation epochs to reach a configuration that allows for backward movement, enabling the robot to move away from the wall. From the second video, we can see that by epoch 50, the robot can turn right more often. Although the right turns may be imperfect and result in a collision, the robot doesn't stay stuck for long. In the last video, at epoch 100, movements and turns are smoother and collisions are reduced. This progression example highlights the effectiveness of the online adaptation mechanism in shaping and refining behavior over time.

Conclusions

This thesis explored online adaptation strategies for robots performing a continuous collision avoidance task through a spectrum of varied adaptive control mechanisms, ranging from Braitenberg-inspired mappings to recurrent neural network (RNN) controllers, with a focus on their ability to adapt in real-time and maintain long-term performance under different conditions.

The experiments carried out demonstrate that even simple reactive architectures can result in effective avoidance behaviors when tuned appropriately, despite suffering from rigidity and limited expressiveness. Neural networks, in contrast, offer greater flexibility and performance, even in the presence of noisy sensors or actuators.

Among all the mechanisms tested, those that retained past memory and used an epoch-based exploration rate yielded the most consistent and high-performing results. These mechanisms achieved rapid convergence, high fitness medians, and low variance, even in the presence of sensor and wheel noise, or when tested in more complex scenarios involving dynamic obstacles. In contrast, other strategies, such as fixed ϵ-greedy (e.g., $\epsilon = 0.1$) or purely memoryless approaches, showcased some limitations. They were more sensitive to noise, slower to reach full success across replicas, or prone to more erratic behavior due to the lack of historical context or poor balance between exploration and exploitation. The worst-performing variants often failed to reach the success threshold consistently or exhibited wide fitness variance, particularly under perturbations or in multi-robot settings.

The study also included a real-world demonstration using a physical Thymio robot, where one of the adaptive RNN controllers was successfully deployed, demonstrating that it is indeed feasible to translate adaptive behaviors from simulation to physical hardware and obtain satisfying results.

Future additional research on the topic could investigate the scalability of the presented approaches to other tasks, such as homing or goal-directed navigation, as well as to more complex tasks involving sequential goals or team-based coordination. Another direction could also include exploring the combination of multiple adaptive mechanisms, aiming to balance the complementary strengths of different strategies. Ultimately, further experimentation and in-depth analysis of the transferability from simulation to reality could help enhance the applicability of adaptive controllers in dynamic, real-world environments.

Appendix A

Custom extensions to the `resilient-swarms/thymio` library

Although these additions were not ultimately used in the final experiments, as part of the early stages of this thesis project, I developed extensions to the resilient-swarms/thymio[1] library to support more behaviors within this Thymio simulation library.

To enable spatial awareness for the Thymio robot in simulation, I implemented a custom positioning sensor plugin that provides absolute position and orientation data. This involved creating both a control interface and a corresponding simulator plugin that reads the robot's position from the ARGoS `EmbodiedEntity` and exposes it to the Lua scripting environment. In Lua, this allowed access to the robot's position and orientation through functions, supporting use cases like gradient sensing, path tracing, or implementing behaviors based on absolute orientation (e.g., taxis or homing). To evaluate the positioning sensor in practice, I tested it through an obstacle avoidance behavior using both Lua and C++.

Building on the positioning sensor, I implemented a synthetic temperature gradient centered at a fixed source location. The simulated temperature was computed as a function of distance from the source, with temperature values that asymptotically approached a maximum near the source. A physical sensor did not measure this temperature; instead, it was derived from the robot's position,

[1] https://github.com/resilient-swarms/thymio

mimicking biological thermal sensing. This sensor was tested on a thermotaxis task where the robot steers toward the heat source using its absolute orientation and position.

Bibliography

[BBR24] Michele Braccini, Paolo Baldini, and Andrea Roli. An investigation of graceful degradation in boolean network robots subject to online adaptation. In Marco Villani, Stefano Cagnoni, and Roberto Serra, editors, *Artificial Life and Evolutionary Computation*, pages 202–213, Cham, 2024. Springer Nature Switzerland.

[Bra86] Valentino Braitenberg. Vehicles: Experiments in synthetic psychology. *Philosophical Review*, 95(1):137–139, 1986.

[BRB23] Paolo Baldini, Andrea Roli, and Michele Braccini. On the performance of online adaptation of robots controlled by nanowire networks. *IEEE Access*, 11:144408–144420, 2023.

[CCM14] Antoine Cully, Jeff Clune, and Jean-Baptiste Mouret. Robots that can adapt like natural animals. *CoRR*, abs/1407.3501, 2014.

[FK10] Dario Floreano and Laurent Keller. Evolution of adaptive behavior in robots by means of darwinian selection. *PLoS biology*, 8:e1000292, 01 2010.

[PTO+12] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, Dec 2012.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.