ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

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

A Comparative Study of Reinforcement Learning Mechanisms for Robot Online Adaptation under Progressive Damage

Master's Thesis in:
Intelligent Robotics Systems

Supervisor
Andrea Roli

Candidate
Valentina Pieri

IV Session
Academic Year 2024-2025

Abstract

This thesis investigates the effectiveness of reinforcement learning mechanisms for online behavioral adaptation in autonomous mobile robots subjected to progressive sensor degradation. Using the ARGoS robotic simulator and a footbot equipped with proximity sensors and differential drive actuators, we examine how four exploration-exploitation strategies enable a neural network controller to maintain an obstacle avoidance behaviour under varying degrees of sensory failure. Controllers are initialized through the use of offline evolutionary optimization via a genetic algorithm (GALib), providing a population of high-performing recurrent neural network configurations as a starting repertoire for subsequent online adaptation. Experiments are then conducted in an online setting across seven damage scenarios, progressively disabling between one and seven of the robot's proximity sensors. Performance is evaluated along two primary axes: a fitness function combining forward velocity with obstacle proximity, and collision counts across adaptation phases. Overall, the findings highlight both the potential and the limitations of lightweight online adaptation mechanisms in dynamic and fault-prone robotic environments.

Contents

List of Figures

List of Tables

Introduction

The capacity for autonomous robots to maintain functional behavior in the presence of unexpected hardware failures represents one of the central challenges in modern robotics, particularly in deployment contexts where human intervention is unavailable or prohibitively costly. Traditional robotic systems, designed around fixed control architectures and predefined behavioral repertoires, are inherently brittle when confronted with degraded operational conditions. Sensor malfunction, whether gradual, through drift and calibration loss, or sudden, through physical damage, presents a particularly insidious type of failure, because unlike mechanical breakdowns, which tend to produce immediately visible behavioral anomalies, sensor degradation may produce misleading signals that can corrupt decision-making in subtle ways.

Reinforcement learning offers a principled framework for online behavioral adaptation, enabling an agent to revise its policy through direct interaction with the environment. The multi-armed bandit algorithms provide an adequate starting point for studying this tradeoff and its variants, including ϵ-greedy policies, Boltzmann action selection, and confidence-bound methods, offer a range of strategies with distinct theoretical properties and practical trade-offs. This work builds upon prior research in online adaptation for mobile robotics, extending that comparative study on exploration mechanisms by specifically analysing the fault-tolerance setting. We investigate how four representative reinforcement learning mechanisms behave when deployed on a footbot robot, whose objective is to roam an arena and avoid collisions with obstacles, with its proximity sensor that are progressively damaged, using the ARGoS multi-robot simulator as the experimental environment. To achieve this a hybrid offline-online architecture is adopted, where neural network controllers are first evolved through a genetic algorithm to establish

a high-quality initialization repertoire, after which online adaptation mechanisms are invoked to refine behavior in the presence of sensor failures.

Structure of the Thesis The structure of this thesis is as following:

- **Chapter 1 - Theoretical Background**: Outlines the theoretical foundations of robotics, neural network controllers, and reinforcement learning, with particular focus on the exploration-exploitation tradeoff and fault-tolerance adaptation.
- **Chapter 2 - Preliminary Experiments**: Presents the experimental setting, the network selection procedure, and an initial robustness analysis under noise and sensor damage using a two-sensor architecture combined with offline evolutionary optimization.
- **Chapter 3 - Analysis with 7 sensors**: extends the architecture to seven proximity sensors and conducts a systematic comparison of four adaptation mechanisms, decaying ϵ-greedy, Softmax, UCB, and VDBE, across progressive sensor failure scenarios.
- **Chapter 4 - Robustness to Penalty**: evaluates the impact of a penalty-augmented fitness function and early stopping on collision control across all mechanisms and damage conditions.
- **Chapter 5 - Thymio Implementation**: Shows the use of one of the proposed mechanisms on a physical Thymio robot, highlighting its feasibility in a real-world setting.

Chapter 1

Theoretical Background

In the last decades, robotics has seen many changes, and so we can define it as a fairly young discipline whose main goal is to achieve the creation of machines that can independently move and decide. However, from when civilization existed, the human mind has sought to create an intelligent being. Great examples of this can be found in literature from Aristotle, arriving to Mary Shelley's most famous character, Frankenstein's creature, and extending to reality with the concept of the automata, objects identified as machines or mechanisms that are relatively self-operating. We can see them as the ancestors of the machines we identify the term with, used prevalently as a substitution for humans for speeding production. These traditional robot systems are typically designed with fixed architectures and predefined behaviors that can effectively operate under specific environmental conditions.

From the 1940s, when the foundation of modern robotics was laid, referred to at the time with the term cybernetics, the field began its rise to popularity with Turing and Wiener, and the concept of computational intelligence. To achieve this objective, many began looking at the natural world from a behavioral perspective to reproduce its mechanisms and adaptive capabilities. This bio-inspired approach led to the development of early control systems and feedback mechanisms that mimicked natural processes, including the pioneering work on artificial neural networks by McCulloch and Pitts in 1943 [MP43], which modeled neurons as computational units. The following decades saw remarkable and different

progress: the 1960s brought the first industrial robots into manufacturing plants and Rosenblatt's Perceptron [Ros58], while the 1970s and 1980s introduced more sophisticated sensors and microprocessor-based control systems. Notably, in 1984, Braitenberg published *"Vehicles: Experiments in Synthetic Psychology"* [Ste85], demonstrating how complex behaviors could emerge from simple sensor-motor connections, profoundly influencing the field. The resurgence of neural networks in the late 1980s with backpropagation algorithms opened new possibilities for robot learning and perception. By the 1990s and 2000s, advances in artificial intelligence and machine learning began transforming robots from purely programmed machines into adaptive systems capable of learning from experience and responding to unpredictable environments.

Neural networks have become a fundamental concept in robotics, enabling the development of adaptive behaviors by processing sensory information, in which simple computational units combine their outputs to produce complex behavioral patterns. This approach mirrors biological nervous systems, where a form of general decision making (thinking) emerges from the collective activity of interconnected neurons rather than from explicit programming. In evolutionary robotics, these neural network controllers evolve, instead of being trained, with architectures structured with input layers activated by the robot's sensors, hidden neurons for information processing, and output layers that control the motors. The weights connecting all these layers encode the robot's behavior and form the genome that undergoes evolutionary optimization. The genome is a sequence of characters whose translation into a phenotype can assume various degrees of biological realism. In *"Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection"* of Floreano and Keller [FK10], refer to this concept that within a population each individual has a distinct genome that can translate to different interactions with the surrounding environment. Given that each of these behaviors affects the robot's fitness, a quantitative measure to evaluate the ability to successfully satisfy its objective.

While evolution approaches enhance robot behaviors across generations of populations, an alternative paradigm emerged from the machine learning community that focuses on how a single agent can learn through direct interaction with its environment: **reinforcement learning**. This approach shares conceptual similar-

ities with evolutionary robotics by relying on trial-and-error processes and the use of a metric (fitness in evolution, reward in RL) to guide improvement. However, whereas evolutionary algorithms evolve populations of controllers over generations, reinforcement learning enables a single robot to progressively refine its behavior through accumulated experience within its lifetime.

Inspired by principles from behavioral psychology and animal learning theory, reinforcement learning conceptualizes the robot as an agent that aims to maximize cumulative reward by identifying which combinations of actions and states lead to desirable outcomes. Through iterative interaction with its environment, the agent progressively refines its policy and, guided by evaluative feedback, incrementally improves its performance. This paradigm has demonstrated particular effectiveness when integrated with neural networks, giving rise to deep reinforcement learning methods that have achieved state-of-the-art results in complex robotic domains—ranging from manipulation to locomotion by learning control policies directly from high-dimensional, raw sensory inputs.

In *"Reinforcement Learning: An Introduction"* by Sutton and Barto [SB98], the reinforcement learning framework is shown to encompass problems of varying complexity, from elementary stateless decision-making to sequential decision processes in dynamic environments. A central challenge discussed in their work is that, in order to achieve high long-term reward, an agent must both select actions that have previously yielded favorable outcomes and continue to search for alternative actions that may produce even higher returns. To discover such potentially superior actions, the agent must engage in exploration, whereas to capitalize on its existing knowledge, it must engage in exploitation. This tension is formalized as the **exploration–exploitation** dilemma, wherein the agent must strike an appropriate balance between acquiring new information about the environment (exploration) and utilizing its current knowledge to maximize immediate reward (exploitation).

This trade-off is most transparently illustrated by the Multi-Armed Bandit problem, a canonical and highly simplified reinforcement learning setting in which an agent repeatedly selects from a set of actions, or "arms," each associated with an unknown reward distribution, with the objective of maximizing cumulative reward over time. In contrast to full reinforcement learning problems, the Multi-Armed

Bandit formulation does not involve state transitions; each decision is conditionally independent of past actions except through the agent's updated beliefs. This absence of state dynamics makes the Multi-Armed Bandit an ideal testbed for isolating and rigorously analysing exploration strategies.

The ϵ-greedy action selection provides a widely adopted solution to the exploration-exploitation tradeoff in multi-armed bandit problems. Under this policy, the agent selects the action with the highest estimated value with probability $(1 - \epsilon)$, exploiting its current knowledge, and chooses a random action with probability ϵ exploring new alternatives. The exploration parameter ϵ can be held constant to maintain perpetual exploration, or more commonly, gradually reduced over time to shift from exploration to exploitation. The decay implements a natural progression from early broad exploration, when the agent knows little about action values, to late-stage exploitation, when sufficient information has been gathered. Despite its simplicity, this algorithm has proven remarkably effective, extending beyond multi-armed bandits to serve as the exploration strategy in more complex reinforcement learning algorithms. While ϵ-greedy addresses exploration in stateless bandit problems, most robotic applications involve sequential decision-making where current actions influence future states and opportunities. These problems are formally modeled as Markov Decision Processes (MDPs), which assume that the future depends only on the current state and action, enabling tractable learning algorithms. Q-learning, introduced by Watkins and Dayan in 1989 [WD92], provides a powerful model-free method for learning optimal policies in MDPs without requiring explicit knowledge of transition dynamics or reward functions. The algorithm maintains a function $Q(s, a)$ whose objective is to estimate the expected cumulative reward obtainable by taking an action a in state s and afterward choosing the most suitable action. At each time step, this Q-value is calculated according to:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big],$$

where:

- α is the learning rate.
- γ is the discount rate parameter.

- R_{t+1} is the receiving reward.
- S_{t+1} is the next state.

The whole second term inside the brackets represents the temporal difference error (TD), the discrepancy between the current Q-value estimate and the target value computed as the immediate reward plus the discounted value of the best action in the next state. Q-learning learns about the optimal policy regardless of which one the agents follow during learning, allowing them to use epsilon-greedy for exploration while learning optimal behavior values. This is referred to as off-policy, and its combination with epsilon-greedy enables systems to discover effective behaviors while gathering best rewarding values.

These systems are very useful tools when applied to an environment where adaptation is needed, where the robots modify their behavior according to immediate feedback from their surroundings. This type of setting is called online adaptation, and it helps in better representing non-stationary environments that change dynamically or situations where the robots experience unexpected events. While online offers more robustness to unforeseen circumstances and aids in shortening the bridge between the virtual and real world, they still face constraints.

On the other side, there is offline adaptation, whose concept links to evolutionary robotics, by being a virtual environment where controllers train before being deployed in the real world. The advantages lie in having robots that are already optimized for their intended task with simulation done. Q-learning itself is most naturally employed in this offline setting, given the slowness of its learning process, while its proven effectiveness suggests that it can inform fully online strategies, as in the case examined in this thesis, where the robot already masters its primary task but finds itself in a slightly altered operational condition. However, this approach assumes that simulation accurately captures real-world physics and that operational conditions match those anticipated during training, assumptions that are frequently disproved.

In *"Robots that Can Adapt Like Animals"* by Cully et al. [CCTM15], the authors address the temporal limitations of purely online adaptation through a hybrid methodology that integrates offline and online processes. Their Intelligent Trial-and-Error algorithm constructs, in an offline phase, a high-dimensional be-

havioral repertoire that maps candidate behaviors to their expected performance. When damage occurs, a Bayesian optimization procedure is employed online to search within this repertoire, selectively testing behaviors predicted to be effective under the altered conditions. Using this approach, a six-legged robot successfully adapted to multiple forms of leg damage in under two minutes, requiring fewer than ten trials. This demonstrates that combining extensive offline preparation with data-efficient online search can achieve both adaptation speed and behavioral flexibility.

These two complementary paradigms, offline and online adaptation, contribute to addressing one of the most challenging problems in robotics: preserving operational capability after damage or partial system failure. The integration of pre-computed knowledge with real-time learning is crucial for autonomous fault tolerance, particularly in scenarios where robots must rapidly compensate for unforeseen malfunctions while operating in degraded states.

In remote or high-risk environments such as space missions, deep-sea operations, or disaster response settings, maintaining functionality following unexpected mechanical failures or component malfunctions is not merely advantageous but often essential, especially when direct human intervention is infeasible or prohibitively costly.

Bongard et al. *"Resilient Machines Through Continuous Self-Modeling"* [BZL06] demonstrated a distinct yet complementary paradigm of damage recovery based on continuous self-modeling, using a four-legged "starfish" robot. Their system maintains an internal simulation of the robot's morphology and employs it to predict behavior under candidate control strategies. When observed behavior diverges from the internal model's predictions, an offline evolutionary algorithm updates the internal model to infer and diagnose the underlying damage. Once the revised model accurately reproduces observed behavior, it is used to synthesize a compensatory gait. In experiments where a leg was physically removed, the robot rapidly re-evolved its internal model and discovered a novel tripod gait within minutes, exemplifying recovery from unanticipated damage by formulating fault tolerance as an inference and model-identification problem.

Collectively, these contributions established two principal and complementary lines of research: model-based approaches that maintain and update internal state

representations, and model-free approaches that discover compensatory behaviors through direct interaction with the environment. While prior work has primarily focused on mechanical damage, sensor degradation has received comparatively less attention. Sensor failures may emerge gradually via drift or abruptly through catastrophic malfunction; unlike many forms of mechanical damage, they can be difficult to detect and may silently propagate erroneous information throughout the control pipeline.

The present work investigates sensor damage recovery using reinforcement learning mechanisms, specifically examining how reinforcement learning mechanisms can sustain obstacle avoidance performance under progressive sensor degradation. By combining offline evolutionary optimization with online reinforcement learning–based adaptation, this approach aims to bridge classical fault-tolerance strategies and contemporary learning-based methods, thereby advancing resilient perception and control in autonomous robotic systems.

Chapter 2

Preliminary Experiments

This chapter introduces the robot, the environment, and the task employed in this study, along with a set of preliminary experiments designed to support the subsequent analysis phase

This research focuses on a small-scale robot, known as a footbot, which is commonly employed in swarm robotics experiments. In this study, a single footbot is utilized within the simulated, modular, and parallel environment ARGoS [PTO+12], which was selected due for its efficiency and flexibility in accommodating diverse experimental setups. The footbot is a ground-based robot that navigates using a combination of wheels and tracks, and is equipped with various sensors and actuators that enable interaction with its environment, usually sensors are provided with read-only access to the simulated environment, whereas actuators are permitted to modify it. The components of the footbot are illustrated in Fig.fig. 2.1.

In this study, only the proximity sensors and wheel actuators are employed. For the proximity sensors, the configuration follows the approach described in *Mechanisms for online adaptation in robots: A comparative study* by Pacilli [Pac25], utilizing 6 out of the available 24 sensors. Each virtual proximity sensor is computed by averaging three physical sensors, with three located on the robot's left side and three on its right. The nearer the footbot is to a surface, the bigger the value generated from the proximity sensors will be. The range provided for the value is $[0, 1]$, where the lower bound is when there are no obstacles in the vicinity,

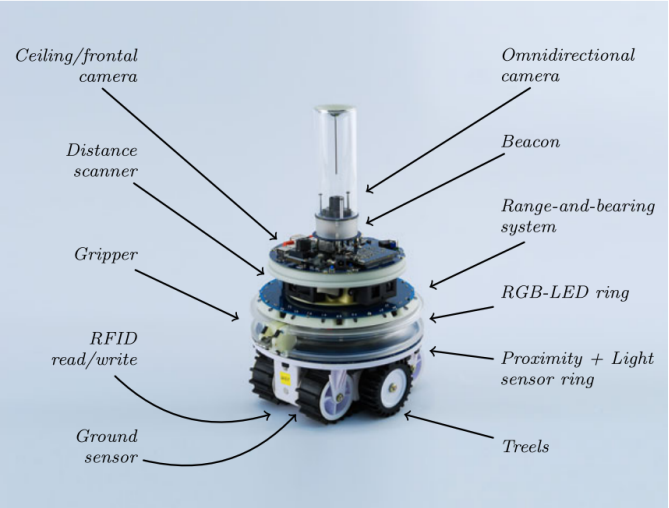


Figure 2.1: The devices composing footbots supported by ARGoS

and the upper bound is when a collision has been made with a surface. Pacilli aimed to replicate the Braitenberg vehicle structure; however, unlike traditional hard coded Braitenberg vehicles, the sensor-to-wheel mapping in this work evolves according to online behavioral adaptation methods.

The simulated environment, where we conducted the experiments, has dimensions of $7 \times 7 \times 1$ units, providing sufficient space for navigation while constraining the task complexity to remain tractable. Five fixed obstacles are placed at predefined points in the arena, introducing navigation challenges that demand active sensing and motor control. The arrangement of these obstacles is chosen to strike a balance between difficulty, adding enough barriers to require avoidance maneuvers, and practicality, avoiding such a high density that no workable navigation strategy would be possible.

We begin our experiments by scrutinizing the best-performing mechanism used by Pacilli, derived from the paper *Reinforcement Learning: An Introduction* by Sutton and Barto [SB98]. The basic logic of it refers to **Recurrent Neural Networks** where, instead of a fixed or parametrically tunable sensor-to-motor mapping encoded in a compact vector, like in the Braitenberg vehicles, the robot's decisions emerge from a layered neural architecture transforming sensory inputs through weighted connections, integrating past internal states. These weights are the parameters of the neural network, which are optimized over time and are en-

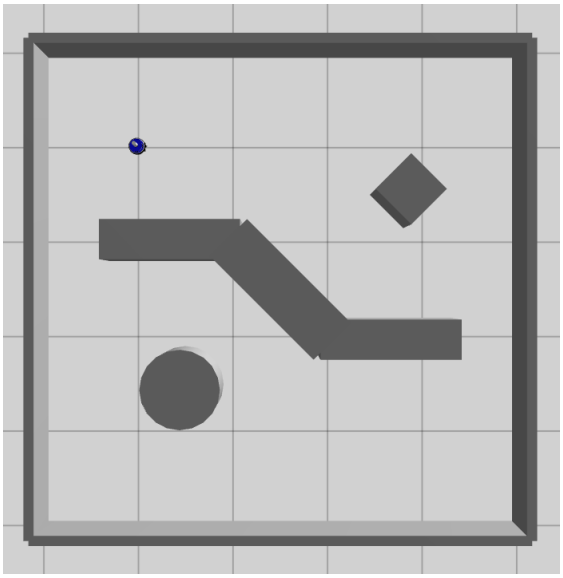


Figure 2.2: Arena of the experiments

coded in θ. In the execution, we run 500 epochs to observe how the footbot's behavior improves in the environment. At each epoch, we have 5000 steps, referred to as `MOVE_STEPS`, where we gather input sensor data and, depending on the configuration used, certain outputs are obtained. At each epoch, the robot senses the environment and, through the footbot sensors, produces two values s_L, s_R, that are then given into the NN, which will output left and right wheel velocities based on the current θ being used. In θ, the network parameters are vectors of real values, of chosen size, referred to as `PARAMS`, initialized with random values in the range $[-1, 1]$, and each vector included is referred to as a configuration.
Two main cases are considered:

- **Zero hidden neurons:** each output unit accumulates a weighted sum from both proximity sensors, a bias term, and a recurrent loop term from its own previous output. The total number of parameters is calculated with:

$$PARAMS = I \cdot O + O + O, \tag{2.1}$$

  where I stands for the input sensors, and O for output wheel actuators. The first O term represents the input-to-output weights, the second O represents

the output biases, and the final one is the recurrent self-loop weights for each output neuron, α_L, α_R. From observing the Algorithm algorithm 1, we can see that the final value is passed through a sigmoid, producing a normalized output in the range $[0, 1]$ that is scaled to yield the commanded wheel velocity.

- **One or more hidden neurons:** for each hidden unit j, the weighted contributions of the sensory inputs and a bias are combined with two recurrent contributions: a self-feedback weight loop and a cross-partner weight loop from a matched neuron $\bar{j}$. The partner index is chosen as the adjacent index $j + 1$ if j is even, $j - 1$ if odd. In this case, the total number of parameters is calculated with:

$$PARAMS = I \cdot H + H + H \cdot 2 + H \cdot O + O, \tag{2.2}$$

  Respectively, the terms represent: input-to-hidden weights, hidden biases, loop and cross-loop recurrent weights per hidden neuron, hidden-to-output weights, and output biases. Hidden activations are passed through the logistic sigmoid to form a vector. Output units are then computed as weighted sums over the hidden neuron, plus biases, followed by the logistic nonlinearity and scaling to physical velocity, as observed in Algorithm algorithm 1.

In our experiments, we used 10 hidden neurons, because in *Mechanisms for online adaptation in robots: A comparative study*, this number of hidden neurons offered the best results in comparison to other values. In Fig. fig. 2.3, there is the graphic representation of the neural network setup with 10 hidden nodes for the footbot.

The resulting motion translates into a behavior that is maintained throughout an entire epoch and evaluated at each step.

The mechanism we first selected for our study is a *Multi-Armed Bandit ϵ-greedy* online adaptation described formally as Algorithm 17 in Pacilli's paper [Pac25]. Among the Multi-Armed Bandit variants examined, epoch-decaying ϵ yielded the best and most stable cumulative performance, typically showing higher medians,

Algorithm 1 `RNN_forward`: Recurrent Neural Network logic

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

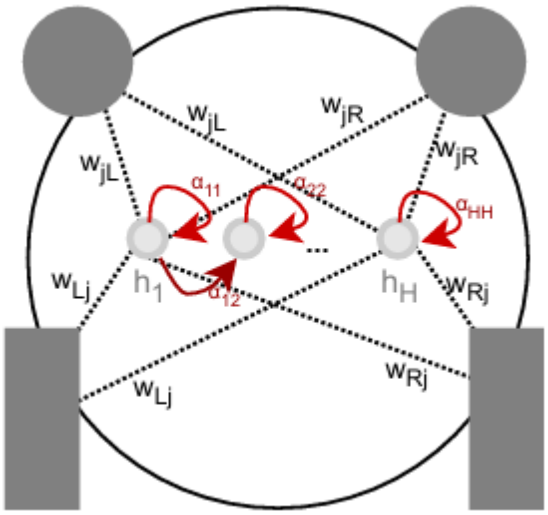


Figure 2.3: Schematic representation of the general feed-forward NN controller with a hidden layer ($H > 0$)

tighter IQRs, and a robust compromise between rapid learning and stable final performance.

The algorithm treats each controller instantiation θ as an "arm" whose expected reward must be estimated from epoch-wise evaluations. Each selection between reusing a promising configuration, exploitation, and perturbing the current configuration to obtain new candidates, exploration, is driven by an ϵ-greedy rule.

At each epoch, the active configuration θ is executed for a fixed number of control steps, `MOVE_STEPS`, and its reward r is calculated at each step using:

$$\Phi_t \;=\; V_t\,(1 - i_t), \tag{2.3}$$

where V_t is the mean of the velocities of the two wheels at time, or step, t. i_t is the maximum value from the sensors at step t, and represents the footbot's level of proximity to obstacles. All signals are normalized to $[0, 1]$. This formula is inspired by the one used by Floreano and Keller in *Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection*, but has been adapted to the footbot's ability to move only forward, with velocity in the range $[0, 10]$. [FK10] In Algorithm algorithm 2 its referred as `evaluate_performance()`.

At the end of each epoch, the **exploration-exploitation** protocol is implemented. The fitness values obtained at each step, which have been accumulated during the epoch, are averaged at the end of it. The epoch reward is the time-average over the epoch, where `T = MOVE_STEPS`:

$$r \;=\; \frac{1}{T}\sum_{t=1}^{T}\Phi_t, \tag{2.4}$$

This measure rewards forward velocity while penalizing proximity to obstacles, and we refer to it as objective function, average fitness, and f_{Obj}. After observing r, the algorithm updates the running value estimates $Q[i]$ for each configuration i. Let Θ denote the set of configurations instantiated during a run. For each $\theta\_i \in \Theta$ the mechanism maintains:

$Q[i]$ empirical value estimate of θ_i, $N[i]$ count of epochs where θ_i was used.

When configuration i yields reward r at an episode, the sample-average update is:

$$Q[i] \leftarrow Q[i] + \frac{r - Q[i]}{N[i]}, \tag{2.5}$$

where $N[i]$ is the number of times i has been evaluated. If the current θ outperforms the configuration previously found as best, it is retained as the new best, and it could be used as a baseline in the following epochs until overthrown. After updating the Q value for the current epoch, the algorithm decides whether to **exploit** the best configuration seen so far or to **explore** by generating a new configuration through mutation. This decision is driven by a random value $p\epsilon[0, 1]$ 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 - ϵ, select θ_j where j = arg $max_k\ Q[k]$ (**exploitation**);
- With probability ϵ, create $\theta_{i+1} \leftarrow mutate(\theta_i)$ (**exploration**).

The ϵ value adapts to the experiment in the $[0.1, 0.9]$ range based on the current epoch, ensuring a stronger early exploration, gradually shifting toward heavier

exploitation at the end of a run:

$$\epsilon(\text{epoch}) = \max\left(0.1,\ 0.9 - 0.8 \cdot \frac{\text{epoch}}{\text{MAX\_EPOCHS}}\right), \tag{2.6}$$

which biases the system towards high exploration early and increased exploitation later, in Algorithm algorithm 2 is referred as ϵ_decay(). In the case of the exploration, the mutation perturbs a single randomly selected parameter in θ and modifies it by adding a small random δ in the range of $[-0.5, 0.5]$. The newly obtained configuration will be used and consequently evaluated in the following epoch.

2.1 Selection of networks

Following the implementation of the selected mechanism, algorithm 2, the subsequent step consisted in identifying the best-performing neural network controllers. This was achieved by applying a targeted selection procedure to candidate networks that met a predefined numerical performance criterion, initially set to a fitness value greater than or equal to `0.40`. To avoid redundant selections, candidate weight vectors were compared via a normalized Euclidean distance, thereby preventing duplicate or near-duplicate configurations from being retained.

In summary, all simulations were conducted in the ARGoS simulator, employing controllers implemented as neural networks with $H = 10$ hidden neurons. The main experimental protocol used `5,000` simulation steps per evaluation and a maximum of `300` training epochs (`MOVE_STEPS = 5,000`, `MAX_EPOCHS = 300`).

A dedicated Python toolchain was developed to support data collection and analysis. One program recursively parsed execution logs and extracted candidate weight vectors; two additional scripts evaluated the best configurations, one operating in batch mode (without a graphical interface) and the other using a graphical interface for visual inspection. A merging script aggregated results across multiple runs, removed duplicates, computed descriptive statistics for the candidate set, and computed a composite score for ranking configurations.

During the validation phase, each candidate configuration was re-evaluated in headless mode (without graphical rendering) for `5,000` steps while additional

Algorithm 2 Online adaptation - ϵ-greedy

Init: Q, $N \leftarrow$ arrays for value estimates and counts
$\theta_i \leftarrow$ randomly initialized configuration
$Q[i] \leftarrow 0,\ N[i] \leftarrow 0$
for epoch **to** MAX_EPOCHS **do**
    $fitness\_accum \leftarrow 0$
    for step **to** MOVE_STEPS **do**
        sense environment $\rightarrow (s_L, s_R)$
        $(outL, outR) \leftarrow$ RNN_forward(s_L, s_R, θ_i)
        set motor speeds using outputs
        $fitness\_accum \leftarrow fitness\_accum +$ evaluate_performance()
    end for
    $f_{Obj} \leftarrow \frac{\texttt{fitness\_accum}}{\texttt{MOVE\_STEPS}}$
    $N[i] \leftarrow N[i] + 1$
    $Q[i] \leftarrow Q[i] + \dfrac{f_{Obj} - Q[i]}{N[i]}$
    $p \leftarrow$ random uniform in $[0, 1]$
    function ADAPTIVE ϵ
        $\epsilon \leftarrow \epsilon$_decay()
        if $p \leq \epsilon$ **then**
                                                ▷ Exploration
            $\theta_{i+1} \leftarrow$ mutate$(\theta_i),\ Q[i+1] \leftarrow 0,\ N[i+1] \leftarrow 0$
        else
                                                ▷ Exploitation
            $\theta_i \leftarrow \theta_j \quad$ where $j = \arg\max_k Q[k]$
        end if
    end function
end for

quantitative metrics were recorded. Only the highest-ranking candidates according to the numerical criteria were subsequently inspected using the graphical interface. The most promising networks, based on both quantitative metrics and qualitative visual assessment, were stored in a final directory, along with their corresponding weight vectors and a log entry reporting average fitness, epoch, number of collisions, and qualitative annotations.

Throughout the selection pipeline, we observed that several configurations exhibited frequent collisions with arena obstacles and barriers. To address this, we introduced a collision counter and refined the filtering criteria to prioritize configurations that combined high fitness with low collision frequency. Among the collected metrics, the collision counter was incremented whenever the readings from proximity sensors exceeded a preset threshold of `0.2`, enabling the automatic rejection of networks that systematically collided with obstacles.

The applied mechanism yielded favorable results: the mean fitness in the earliest epochs was approximately `0.38`, subsequently surpassing the `0.40` threshold and, in the best-performing cases, stabilizing in the range `0.42`–`0.45`.

A substantial number of controller executions were performed, during which actuation noise was injected into one of the two wheels at random. This noise was sampled uniformly from the interval $[-noise, noise]$ and had a magnitude corresponding to 10% of the maximum attainable wheel velocity. Across successive controller runs, the wheel to which the noise was applied was alternated.

From these experiments, we extracted the best-performing parameter vectors (θ), validated them automatically, and subsequently processed them with a merging/aggregation script. This script consolidated the results, removed duplicate solutions, computed descriptive statistics for the candidate controllers, and generated a composite ranking score. Based on a combination of numerical evaluation and qualitative, video-based inspection, thirty neural network controllers were selected as final candidates.

The performance statistics of these 30 selected networks, evaluated over a single epoch of 5,000 simulation steps, were as follows:

- **Collisions (5,000 steps / 1 epoch):** min = 130, max = 302, mean $\approx$ 227.13, median = 216.

- **Average fitness (5,000 steps / 1 epoch):** min = 0.474021, max = 0.484665, mean ≈ 0.479418, median ≈ 0.479343.

A qualitative analysis of the resulting behaviors revealed that several networks exhibited asymmetric motion patterns. In particular, some controllers displayed a systematic bias toward turning either left or right, with a predominance of right-turning behavior. This bias is likely attributable to the typical spatial configuration of the foot-bot within the arena. We hypothesize that, over the course of the simulation, the robot's online adaptation process preferentially reinforced configurations that rotated in that direction, as these produced higher fitness values.

The final set of 30 networks, having been deemed satisfactory according to the combined criterion of quantitative performance and qualitative behavioral assessment, was archived together with their corresponding execution logs for subsequent experimentation and analysis.

This point marked the beginning of a second phase of experimentation with the selected networks, in which we investigated whether their performance could be further improved. We re-ran the online adaptation procedure, increasing the number of training epochs from 300 to 500. The resulting performance metrics were:

- **Collisions (5,000 steps / 1 epoch):** min = 64, max = 164, mean ≈ 124.3, median = 136;

- **Average fitness (5,000 steps / 1 epoch):** min = 0.480321, max = 0.489344, mean ≈ 0.484466, median ≈ 0.48486.

Re-evaluating the best-performing networks under these extended training conditions resulted in an approximately twofold reduction in the number of collisions. Average fitness values also improved, although to a lesser extent and not proportionally to the reduction in collisions, with the median increasing from `0.479343` to `0.48486`.

To estimate the upper performance bound attainable by our networks, we conducted three additional evaluation runs for each of the best 30 networks identified in the previous experiment. The mean fitness typically did not exceed a value of `0.485`, with only a few rare exceptions. A similar saturation effect was observed

for the number of collisions: although the average number of collisions generally decreased below 100 (and often substantially below this threshold), even for those configurations that yielded the lowest collision counts, approaching zero, in some of the 500 evaluation runs, we still observed iterations with approximately 100 collisions.

This variability can be attributed to several factors, potentially acting in combination. First, at every timestep, the controller must compensate for **stochastic perturbations**, introduced as random noise applied to the actuators. Second, part of the non-deterministic behavior can be explained by the continuously **changing initial conditions** at the beginning of each epoch: since we observe the footbot across multiple epochs within the arena, its starting position and orientation differ from epoch to epoch, and configurations that are well adapted to a particular initial condition may not generalize to subsequent ones. Third, some controllers may exhibit an inherent **bias** toward steering preferentially to the right or to the left. Such a directional bias can induce collisions in situations where the footbot should, for instance, turn left to avoid an obstacle but instead tends to turn right, thereby increasing the likelihood of collision events.

2.1.1 Noise

Considering that the data obtained previously were collected under noisy conditions, we elected to reduce the noise level to zero in order to identify an additional 30 networks. These networks served as the baseline for subsequent experimentation, providing robust behavioral characteristics uncompromised by noise-induced perturbations.

Subsequently, we began our testing, to asses the effectiveness of the the networks, identified and optimized in a noise-free condition, under adverse conditions with a degraded operational status. We identified to main categories:

- **Light malfunction.** Noise applied to different components.
- **Heavy malfunction.** Components not working as intended, completely or partially.

Firstly we analyzed the **light malfunction** case, simulated through systematic evaluation under varying conditions characterized by:

- Different `noise` levels: `0.05`, `0.1`, and `0.3`;
- Different `application` targets: `sensors`, `actuators`, and `both`.

Thus, nine distinct experimental conditions were established, each evaluated across 30 networks over 500 epochs. The subsequent analysis presents the quantitative results obtained from these nine noise conditions, utilizing box-plot distributions computed across the 30 candidate networks for each condition. The analysis focuses primarily on two key metrics: average fitness, *fitness objective* (f_{obj}), which measures locomotion efficiency and obstacle avoidance, *collision count*, which quantifies the number of collisions encountered during the epoch.

Collision counts With respect to the **collision distributions**, as illustrated in Figure fig. 2.4, the conditions employing noise levels of `0.05` and `0.1` exhibited substantially homogeneous behavior across all noise sources. All six of these conditions demonstrated moderate variability, with standard deviations ranging from approximately 59 to 63 and medians oscillating between 188 and 196 collisions. The distributions displayed a mild negative skew, indicating a concentration of observations toward the lower end of the inter quartile range. This consistency across both individual noise sources and their combination suggests that the online adaptation mechanism maintained behavioral stability at these intensities, irrespective of the channel through which the perturbation was introduced.

At the noise level of `0.3`, however, the system's behavior diverged markedly depending on the noise source:

- **Sensor** noise, the median collision count decreased to 134 (mean $\approx$ 152.6), representing a substantial reduction relative to all other conditions. This result suggests that elevated levels of sensor noise may paradoxically induce more conservative navigation strategies: perceiving a noisier and therefore more challenging environment, the robot may adopt behaviors that reduce forward velocity, thereby decreasing the frequency of physical contact with obstacles.

- **Both** noise exhibited the highest variability observed across all conditions, with a standard deviation of approximately 100.4 and an inter quartile range of 143. This pronounced heterogeneity indicates that concurrent perturbation of both sensory and actuation channels destabilizes the system to a degree that neither channel alone achieves, producing a more erratic and less predictable behavioral repertoire across the network population.

- **Actuator** noise preserved a moderate level of variability (standard deviation ≈ 58.6), comparable to that observed at lower noise intensities, with a median collision count slightly elevated at approximately 200. The overall distribution remained comparatively compact, indicating that actuator-only perturbations produce more predictable effects on collision dynamics than sensor noise. This is likely attributable to the capacity of the online adaptation mechanism to partially compensate for actuation errors through the feedback loop inherent in the control architecture, whereas corrupted sensory input fundamentally alters the perceptual foundation upon which control decisions are predicated.

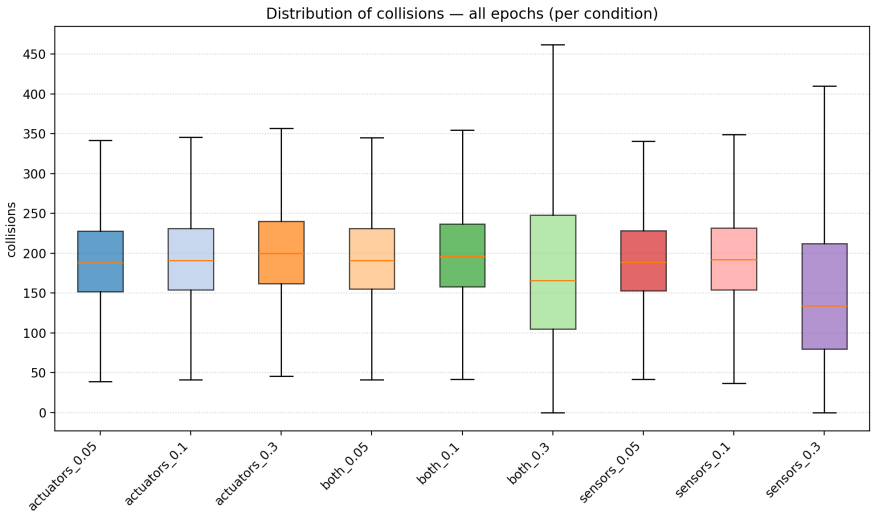


Figure 2.4: Comparison of collision counts across the nine distinct noise conditions.

Average fitness (`fObj`) Concerning the **average fitness**, which aggregates forward velocity and obstacle avoidance into a single scalar performance metric

(Figure fig. 2.5), a markedly different sensitivity profile emerges compared with collision counts. At noise levels of 0.05 and 0.1, the coefficient of variation of `fObj` remained exceptionally low across all six conditions, ranging from 1.31% to 2.11%, with distributions tightly clustered around their respective medians. This minimal dispersion indicates that the online adaptation mechanism produces behaviorally coherent solutions in terms of overall task performance, even when underlying collision dynamics vary across conditions.

At the noise level of 0.3, statistically significant degradation in `fObj` was observed, though its magnitude depends critically on the noise source:

- **Sensor** noise yielded a mean `fObj` of approximately 0.446, representing a degradation of roughly 3.8% relative to the noise-0.05 baseline. In this condition, performance declined below the values attained at lower noise intensities.

- **Both** noise showed a comparable decline to sensor noise, with a mean `fObj` of approximately 0.444, representing a degradation of roughly 4.0%. As with sensor-only noise, this condition produced fitness values below the average of the remaining conditions.

- **Actuator**-only noise is particularly noteworthy: despite the increased collision variability documented above, the mean `fObj` in this condition remained at approximately 0.464, exhibiting a negligible degradation of only 0.1% relative to lower noise intensities. This dissociation between collision count and objective function value under actuator noise indicates that the network architectures selected through the evolutionary pipeline demonstrate partial robustness to actuation perturbations; the controllers maintain sufficient forward velocity to preserve the composite fitness score, even when the mechanical execution of that velocity is subject to stochastic corruption.

The asymmetry observed in the actuator-only condition is consistent with the architectural roles of the two channels within the control loop: sensor noise corrupts the perceptual representation upon which all control decisions are grounded, thereby propagating uncertainty throughout the entire decision-making process,

whereas actuator noise affects only the final execution stage. The online adaptation mechanism, which operates by updating value estimates based on observed rewards, is consequently less capable of compensating for errors arising at the input stage than for those manifesting at the output stage.

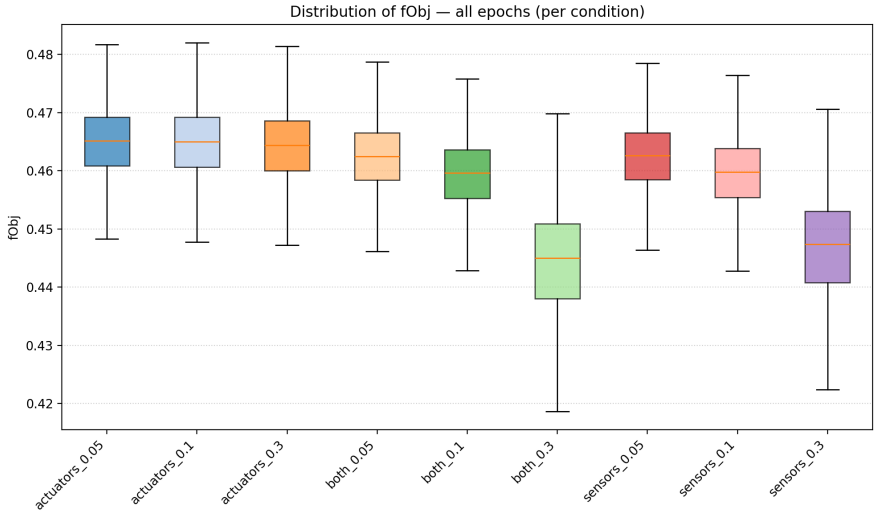


Figure 2.5: Comparison of average fitness (`fObj`) across the nine distinct noise conditions.

A direct comparison across noise types reveals that sensor perturbation, as well as the combined perturbation of sensors and actuators, exerts a substantially more deleterious effect on average fitness than actuator-only perturbations of equivalent magnitude. This asymmetry is consistent with the architectural roles of the two channels within the control loop: sensor noise corrupts the perceptual representation upon which all control decisions are grounded, thereby propagating uncertainty throughout the entire decision-making process, whereas actuator noise affects only the final execution stage. The online adaptation mechanism, which updates value estimates based on observed rewards, is consequently less capable of compensating for errors arising at the input stage than for those manifesting at the output stage.

Scatter plots Figures fig. 2.5 and fig. 2.4 provide comprehensive insight into the overall behavior of the networks under different experimental conditions. To

achieve a more granular understanding, we conducted an in-depth analysis of the data. The objective was to generate, for each of the nine noise conditions, a set of six scatter plots in which the horizontal axis represents the objective function and the vertical axis represents the collision count. To populate these plots, a set of parameters was defined for each network, which gathered `fObj` and `collisions`, for the specific epoch they represented.

- f_0: the original network (trained with noise set to zero) evaluated over a single epoch.
- T_0: the first epoch of the online adaptation process.
- T_1: in contrast to T_0, the final epoch of the adaptation process.
- $T_{\max}$: the epoch that produced the best `fObj` value.
- $\langle V(f_0)\rangle_{100}$: the original network adapted over 100 epochs.
- $\langle V(T_{\max})\rangle_{100}$: analogous to $\langle V(f_0)\rangle_{100}$, but represents the data of the best-performing adapted network.

For all evaluation points except the final two, the objective function and collision count values represent means recorded directly from the corresponding single epoch; for $\langle V(f_0)\rangle_{100}$ and $\langle V(T_{\max})\rangle_{100}$, these values are averaged over the 100 evaluations.

These six evaluation points are then compared pairwise to visualize the effect of adaptation across different stages of the process. The six pairings are as follows:

- f_0-$T_{\max}$: compares the original network against the best epoch produced during adaptation.
- f_0-T_1: compares the original network against the final epoch of adaptation.
- T_0-$T_{\max}$: isolates the improvement achieved during adaptation by comparing its initial and best-performing epochs.
- T_0-T_1: compares the first and final epochs of adaptation to capture the overall trajectory of the process.

- T_1-$T_{\max}$: highlights whether the final epoch of adaptation coincides with or deviates from the best-performing epoch.

- $\langle V(f_0)\rangle_{100}$-$\langle V(T_{\max})\rangle_{100}$: compares the averaged performance of the original and best-adapted networks over a large number of evaluations, thereby reducing the influence of stochastic variability.

Consequently, each scatter plot contains one data point per network for each of the two evaluation points being compared, yielding 30 data points per series. To accompany each plot, a corresponding table is provided to report the exact numerical values of the objective function and collision count at each of the evaluation points involved in the comparison. These tables facilitate direct correspondence between every point displayed in the plots, enabling both verification and detailed inspection of individual network behaviors.

Analysis of the data obtained for the T_0–T_1 combination indicates that the two datasets exhibited highly similar behavior, differing primarily in the ranges of their average fitness values and collision counts, depending on their noise conditions.

- **Sensor** noise: the most performing values were found with noise `0.05`, having the highest `fObj` and lowest collisions; the worst results were noticed with `0.3`.

- **Both** noise: `0.05`, higher number of networks with `fObj` above 0.46 but collisions around 100-250; `0.1`, more networks have collisions lower than 100 but average `fObj` a little lower; `0.3` the worst average `fObj` (0.43-0.46) but is able to achieve collisions under 50.

- **Actuators** noise: all three cases show `fObj` achieving even 0.48; `0.05` and `0.1` offer lower collisions (0-100).

Across all conditions, the T_1–$T_{\max}$ pairing exhibited the greatest divergence in both average fitness and collision count, with $T_{\max}$ consistently yielding superior performance. This makes us notice that the data of the best epoch and the last epoch also differ, meaning that the footbot wasn't able to use its best performing epoch in an efficient way in the following epochs.

- **Sensor** noise: the most performing values were found with noise `0.5`, having the highest `fObj`; the lowest `fObj` results were achieved with `0.3`, but having the average lowest collisions (0-50).
- **Both** noise: similar to the sensor's noise data.
- **Actuators** noise: `0.05` had the lower collisions for T_{max} (0-150), but `0.3` shows the best recovery from T_0 with T_{max}, Fig. fig. 2.6;

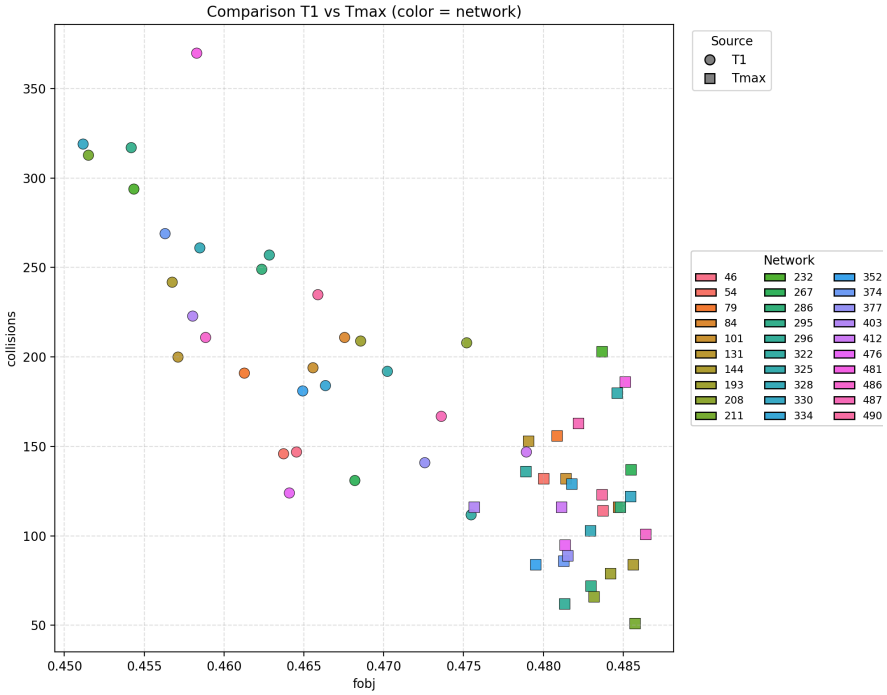


Figure 2.6: Comparison of 30 networks with actuator noise of `0.3`.

2.1.2 Damage

To further investigate the malfunction cases, we considered the second scenario, the one with **heavy malfunctions**, in which the footbot needs to manage **malfunction events** on its components. In our context, two primary categories can be identified: those affecting the **actuators** and those affecting the **sensors**. However, whereas in the **first** case the footbot exhibits locomotion difficulties that

compromise the primary objective of the experiment, namely, the ability to avoid obstacles, the **second** configuration can instead be regarded as an appropriate and methodologically sound experimental setup for detailed analysis. The implementation of the sensor malfunction consisted of constraining the sensor readings to remain constant throughout the entire experiment, thereby causing one of the sensors to be classified as "damaged" for the entirety of the execution. We considered damage levels of `0` and `0.5`, where the sensors could share only either of those values for the whole execution. Two distinct application scenarios were analyzed, in which the damage was imposed separately on the **left** and **right** sensors.

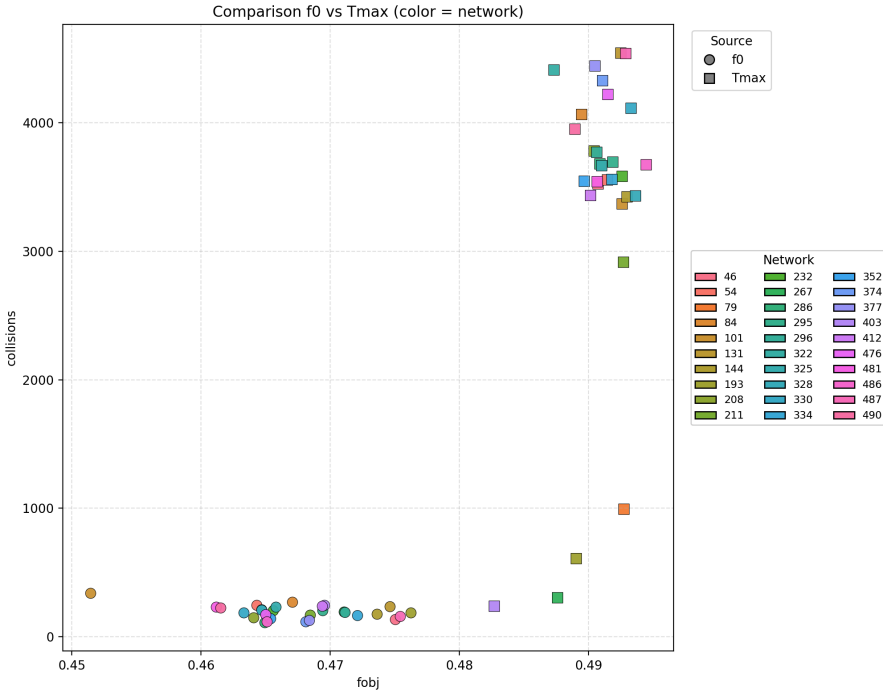
Total damage (`0.0`) Statistical analysis across the 30 networks revealed that under total failure, the online adaptation mechanism not only compensated for sensor loss but achieved fitness improvements relative to pre-damage baselines. For the comparison between $f_0 - T_{\max}$ we an increase for both sensors, where f_0 mean `fObj` started at `0.4674`:

- **Right** sensor: T_{max}, improved of +5.03% with mean `fObj = 0.4909`, Fig. fig. 2.7a.
- **Left** sensor: T_{max}, improved of +4.75% with mean `fObj = 0.4896`, Fig. fig. 2.7b.

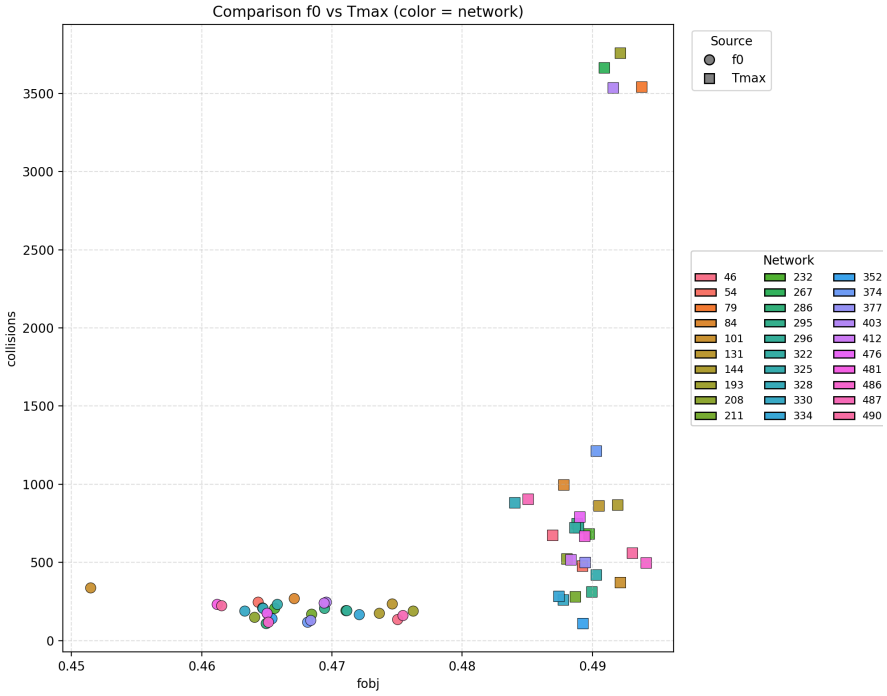
The recovery process demonstrated progressive improvement: while the comparison $f_0 - T_1$ showed minimal changes (+0.56% for **right** sensor with mean of `0.4700` and +0.66%for **left** sensor with mean `0.4705`), the pair $T_1 - T_{\max}$ yielded substantial gains of +4.45%, **right** sensor, and +4.06%, **left** sensor. However, these fitness improvements were accompanied by dramatic increases in **collision** frequency from approximately `193` collisions at f_0 to:

- `3,364` at $T_{\max}$ for **right** sensor damage, Fig. fig. 2.7a.
- `1,012` at $T_{\max}$ for **left** sensor damage, Fig. fig. 2.7b.

Qualitative behavioral observations corroborated these statistical findings while revealing distinct lateral asymmetries. With the **right** sensor damage, the footbot exhibited high collision counts and a pronounced tendency to become stuck against



(a)



(b)

Figure 2.7: Comparison of the 30 networks with total damage `0.0`*, for pair* $f_0 - T_{\max}$*:* `(a)` ***right*** *damaged sensor fig. 2.7a and* `(b)` ***left*** *damaged sensor fig. 2.7b*

obstacles; nevertheless, it retained the capability to navigate around the arena. Conversely, damage to the **left** sensor demonstrated superior adaptive behavior: although the footbot could become temporarily stuck upon contact with surfaces on the damaged side, similarly to its counterpart, it proved more capable of self-extricating and moving away from obstacles, successfully touring the entire arena. This asymmetry aligns with the substantially lower collision count observed for left sensor damage in the statistical analysis.

Partial damage (`0.5`) In stark contrast to total damage, partial sensor degradation produced severe and persistent performance deficits. The comparison $f_0 - T_{\max}$, f_0 mean `fObj = 0.4674`, revealed substantial fitness reductions:

- **Right** sensor: T_{max}, decreased of -16.03% with mean `fObj = 0.3925`.
- **Left** sensor: T_{max}, decreased of -19.83% with mean `fObj = 0.3747`.

While the networks demonstrated substantial recovery at the end of the adaptation, $T_0 - T_1$:

- **Right** sensor: T_0 with mean `fObj = 0.2496`, T_1 improved with mean `fObj = 0.3924`, $+57.23\%$.
- **Left** sensor: T_0 with mean `fObj = 0.2470`, T_1 improved with mean `fObj = 0.3746`, $+51.66\%$.

$T_1 - T_{\max}$ showed slight improvements of $+0.02\%$, **right**, and $+0.04\%$, **left**, indicating that the adaptation mechanism was unable to restore performance to pre-damage levels despite continued exploration.

Qualitative observations under partial damage conditions revealed fundamentally distinct and pathological behavioral patterns. Both sensors damaged at the `0.5` intensity level induced persistent circular motion, wherein the footbot continuously rotated around its own axis rather than executing goal-directed navigation. This phenomenon can be attributed to the systematic distortion of sensory input: the sensors operating at `0.5` provide misleading proximity information that erroneously triggers the obstacle-avoidance reflex, thereby generating continuous

turning behavior. Based on these observations, we established that damage intensity values must remain strictly below the lower activation threshold of the obstacle-avoidance behavior to prevent such maladaptive rotational patterns.

The counterintuitive finding that complete sensor failure yields superior outcomes compared to partial degradation can be attributed to the computational distinction between absent and systematically distorted information: a disabled sensor provides a constant signal that the network can learn to disregard, whereas a sensor operating at reduced capacity delivers misleading proximity data that actively interferes with decision-making processes.

Chapter 3

Offline Adaptation

Considering that the results obtained purely from the online adaptation did not offer noteworthy and meaningful results, we decided to implement an offline simulation framework. This behavior exposed a fundamental problem: despite executing 500 epochs with 5000 steps each, the ϵ-greedy strategy combined with single-parameter mutation proved inadequate for discovering sturdy controller configurations.

The offline simulation addresses this limitation through evolutionary optimization, specifically employing the `GAlib` genetic algorithm library. The genetic algorithm maintains `100` diverse candidate solutions simultaneously, enabling parallel exploration of multiple parameter space regions. This global search dramatically increases the probability of discovering high-performing configurations by exploring many promising areas simultaneously rather than following a single trajectory through the landscape. This comprehensive search provides substantially higher confidence in discovering rewarding solutions within the feasible parameter space.

The framework serves as an initialization mechanism for online adaptation, implementing a hybrid approach that leverages complementary strengths of both paradigms. The evolutionary phase conducts a thorough exploration to identify promising 30 networks through a systematic population-based search, where the mechanism used to evaluate them is by using Floreano and Keller's inspired performance formula, Equations eq. (2.3) and eq. (2.4).

3.1 Architectural Components and Computational Workflow

The offline simulation architecture comprises **four** primary components: the **evolutionary** process orchestrated by GALib, the ARGoS **simulator** that executes trials and provides the simulated environment, **loop functions** that mediate between the optimizer and the simulator, and the neural-network **controller** that computes the trial performance by mapping genome parameters to robot actions.

The evolutionary simulation is orchestrated through the process implemented in Algorithm algorithm 3, which leverages the GALib library to implement a canonical genetic algorithm. The system initializes a real-valued genome of fixed dimensionality with `GENOME_SIZE = 72` parameters, with each allelic value constrained to the interval $[-1, 1]$. This genome encoding represents the parameterization of the neural configuration, encompassing all weights connecting sensor inputs to hidden units, hidden units to motor outputs, bias terms for each neuron, and recurrent connections. The population size of `100` balances diversity against computational cost, while `500` generations, equal to the number of epochs in the online adaptation. The mutation rate of `0.05` introduces variation for escaping local optima while preserving beneficial parameter combinations. The crossover operator, applied with probability `0.15`, synthesizes partial solutions from different individuals in the population, combining complementary beneficial characteristics that single-trajectory methods cannot achieve. When two parent solutions each possess different advantageous features, for instance, one excelling at rapid forward motion and another at precise obstacle detection, crossover can produce offspring that inherit both capabilities.

The `step()` method encapsulates this generational cycle, evaluating all individuals using the `LaunchARGoS` function, recording fitness statistics, and periodically saving the best discovered solution, enabling progress monitoring and result recovery.

The ARGoS configuration file `galib.argos` defines the simulated environment with the parameters matching the online adaptation scenario: a $7 \times 7 \times 1$ arena with five static obstacles, ensuring direct comparability between evolutionary and adaptive approaches.

Algorithm 3 Main Evolutionary Loop

Init: Population size = 100, Generations = 500, Mutation rate = 0.05, Crossover rate = 0.15

Initialize allele within range $[-1.0, 1.0]$
Create Genomes with GENOME_SIZE parameters
Create cGA genetic algorithm with Genomes

Load ARGoS simulator with experiment file "experiments/galib.argos"

while $\neg$ cGA.done() **do**
    reset evaluation counter and fitness storage

    cGA.step() $\rightarrow$ LaunchARGoS(genome)
    collect fitness values

    `gen_index` $\leftarrow \max(0,\ cGA.generation() - 1)$

    log `gen_index` and fitness to `evolution_fitness.dat`
end while

Algorithm 4 `LaunchARGoS`: Genome Evaluation Function

function `LaunchARGoS`(*genome*)

Cast *genome* to `g_real` (real-valued genome type)
Retrieve singleton instance of *Simulator*
Retrieve singleton instance of *Loop_Functions*
`simulator` $\leftarrow$ ARGoS simulator
`loop_functions` $\leftarrow$ `GetLoopFunctions()`

Initialize $f_{Obj} \leftarrow 0.0$
`loop_functions`.SetTrial(1)
simulator.Reset()
`loop_functions`.ConfigureFromGenome(g_real)
`simulator`.Execute() $\leftarrow$ `NN_controller`
$f_{Obj} \leftarrow loop\_functions.Performance()$
Store fitness and increment evaluation index
return f_{Obj}

The interface between the evolutionary algorithm and robotic simulation is implemented through the `LaunchARGoS` function, which serves as the objective function for genome evaluation. Algorithm algorithm 4 presents the complete evaluation protocol. The function first converts the generic genome representation into the specialized real-valued genome type, enabling direct access to parameter values. It then retrieves singleton instances of both the ARGoS simulator and custom loop functions, ensuring consistent state management across all evaluations within a generation.

The function executes a procedure intended to guarantee reproducibility for every genome examination. For each genome, it resets the simulator to establish clean initial conditions, transfers parameters through the loop functions interface, executes the complete simulation episode, and retrieves the resulting fitness. This standardized protocol ensures fairness and reproducibility across all evaluations.

Algorithm 5 Loop Functions: Genome-to-Controller Transfer

function ConfigureFromGenome(g_real)
  Initialize `vec_theta` as vector of size GENOME_SIZE
  for $i = 0$ to GENOME_SIZE -1 **do**
    `vec_theta[i]` $\leftarrow$ `c_genome[i]`
  end for
  controller.SetThetaFromGenome(`vec_theta`)
end function

function Performance
  if *controller* $= NULL$ **then**
    Log error: "Controller pointer null in Performance()" **return** 0.0
  end ifreturn *controller*.GetFinalEvaluation() $\leftarrow$ `evaluate_perfomance()`
end function

The loop functions implement the bridge between GALib and ARGoS through two primary responsibilities: parameter transfer and fitness extraction. Algorithm algorithm 5 presents the last two operations, where at initialization and reset operations, it creates and positions the robot entity within the arena according to predetermined initial conditions. The `ConfigureFromGenome` method extracts parameter values from the genome and installs them in the controller by invoking the method `SetThetaFromGenome`, Algorithm algorithm 6. The `Performance` method

retrieves the fitness computed during simulation, Eq. eq. (2.3) and Eq. eq. (2.4). This clean separation allows the genetic algorithm to manipulate abstract parameter vectors without knowledge of neural network architecture, while the controller receives complete specifications without awareness of evolutionary processes.

3.2 Neural Network Controller Implementation

The Neural Network controller is the code that implements the parameterized obstacle avoidance behavior whose computational structure mirrors that of the online adaptive controller. The controller accepts external parameter specification through the `SetThetaFromGenome` method, Algorithm algorithm 6, initializes the controller for each evaluation by copying parameters, resetting recurrent state variables to zero, and clearing fitness accumulators. This establishes a clean computational state, eliminating carryover effects between evaluations.

Algorithm 6 SetThetaFromGenome: Controller Initialization

function `SetThetaFromGenome(vec_theta)`
vec_theta is of size *param_count*

vec_theta←*vec_theta* ▷ Copy parameter vector

Reset hidden and output unit activations

Initialize fitness accumulator
Reset step counter
Clear final fitness and termination flag

As explained before during execution, the controller implements a logic identical to its online counterpart, as seen in Algorithm eq. (2.6), ensuring behavioral comparability. At each control step, the controller computes virtual sensor readings by aggregating and normalizing proximity sensor data, which serve as inputs to the neural network forward pass, which computes motor commands through either direct sensor-to-motor mappings or through hidden layer transformations. The computed wheel velocities are applied to the robot's differential drive actuators, producing motion in the simulated environment. Simultaneously, the controller

evaluates instantaneous performance using the fitness function and accumulates this value for epoch-level aggregation.

3.3 Experiments

As previously described, the experimental simulation preserves strict consistency with the online adaptation setup: each episode comprises 5000 simulation steps, the arena configuration is identical, and the sensor specifications are matched. By recording the fitness values of all individuals in each generation, we were able to filter and identify the best-performing networks, and, as before, selected the top 30 of them. This way, the genomic evolution provided 30 new networks for subsequent online adaptation. The parameters T_0, T_1, and T_{max}, already introduced in the previous chapter, are also employed in the following experiments, where they denote the fitness at the beginning, at the final epoch, and the best fitness reached during the run, respectively.

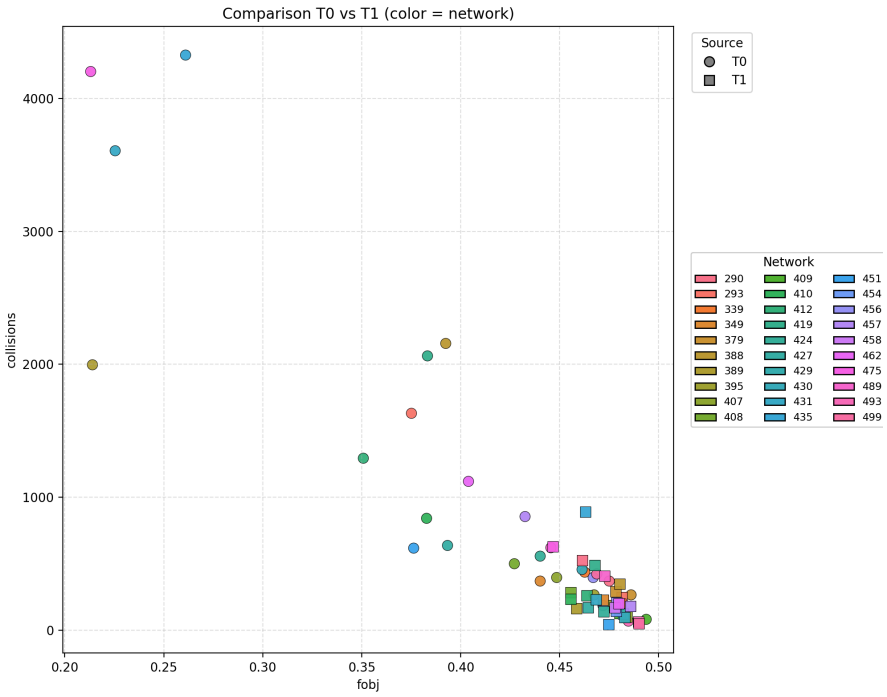


Figure 3.1: Comparison of the 30 networks of the T_0-T_1 pair without damage and threshold 0.2.

Without damage In the first batch of runs with these newly evolved networks, we observed very high final epoch fitness values, T_1, in the range `0.46`-`0.48`, and best epoch fitness values, T_{max}, in the range `0.48`–`0.50`, both with some exceptions. Comparing the initial epoch fitness values, T_0, with the corresponding final epoch values, T_1, across all networks, we observed an improvement both in average fitness and in collision-related metrics, as illustrated in Figure fig. 3.1. However, upon visual inspection performed to validate that the footbot's behavior qualitatively matched the expected task performance, we found that the robot was not behaving as intended.

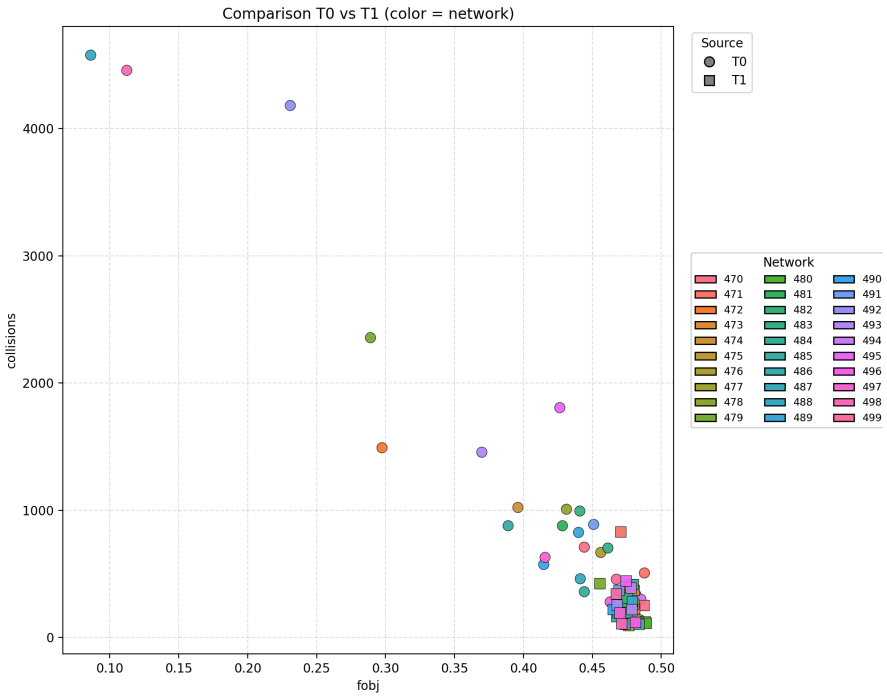


Figure 3.2: Comparison of the 30 networks of the T_0-T_1 pair without damage and threshold 0.1.

A more detailed analysis revealed a critical flaw in our decision to remove the exploration mechanism from the controller. In the original implementation, the `Explore()` function continuously perturbed the wheel velocities at each simulation step, maintaining behavioral variability throughout the episode. By removing this component in both offline and online adaptation, wheel velocities are now initialized only once during setup and remain constant unless obstacles are detected.

Given the footbot's initial distance from obstacles, asymmetric wheel velocities induce a persistent circular trajectory that exploits the arena's spatial characteristics to avoid obstacle contact altogether, preventing activation of the avoidance module. This systematic bias invalidates the resulting fitness measurements, as high values reflect exploitation of this degenerate locomotion pattern rather than genuine obstacle-avoidance competence.

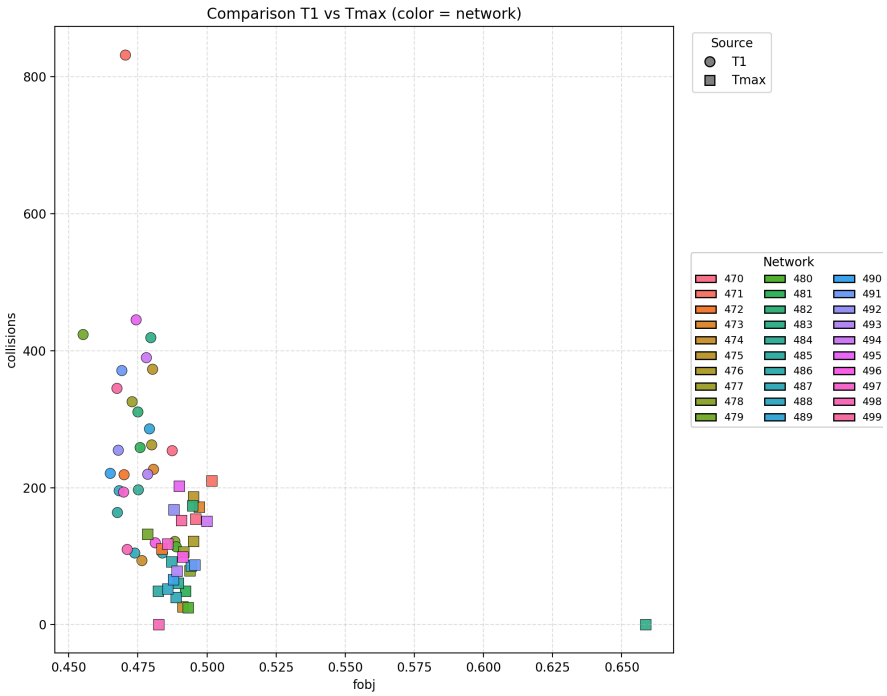


Figure 3.3: Comparison of the 30 networks of the $T_1 - T_{max}$ pair without damage and threshold 0.1.

Following this anomalous behavior, we decided to revert and reintroduce the exploration behavior in the controller, but with a decreased threshold, going from `0.2` to `0.1`.

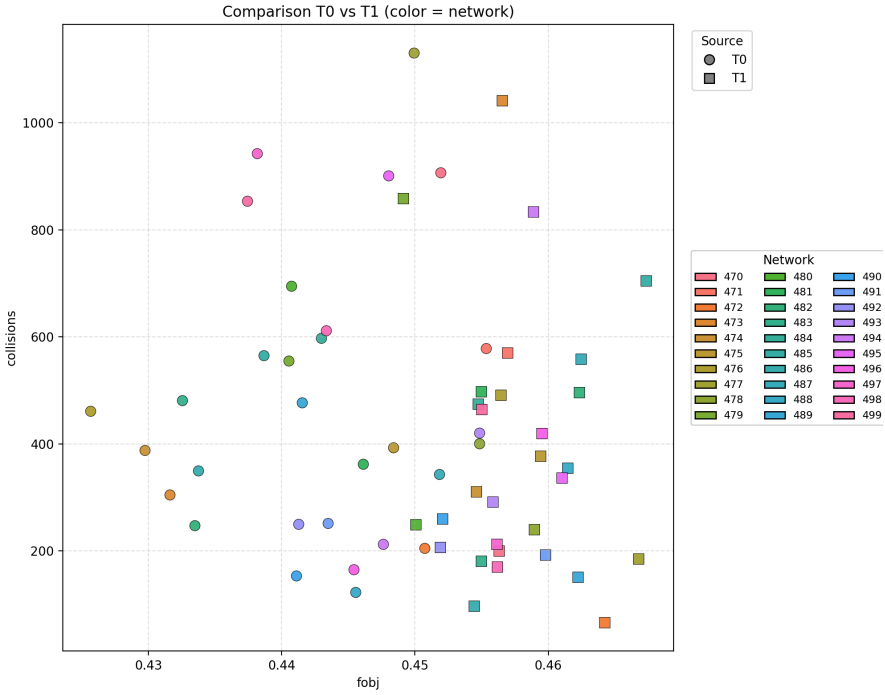
- T_0: mean `fObj = 0.4053` and `coll = 1126`.
- T_1: mean `fObj = 0.4750` and `coll = 265`.
- T_{max}: mean `fObj = 0.4963` and `coll = 101`.

T_0 showed a setback in comparison to f_0, with mean `fObj = 0.4674`, but a pronounced improvement could be seen in T_1 correlated to T_0, Fig. fig. 3.2 and T_{max} achieved even higher `fObj` and `collisions`, Fig. fig. 3.3.

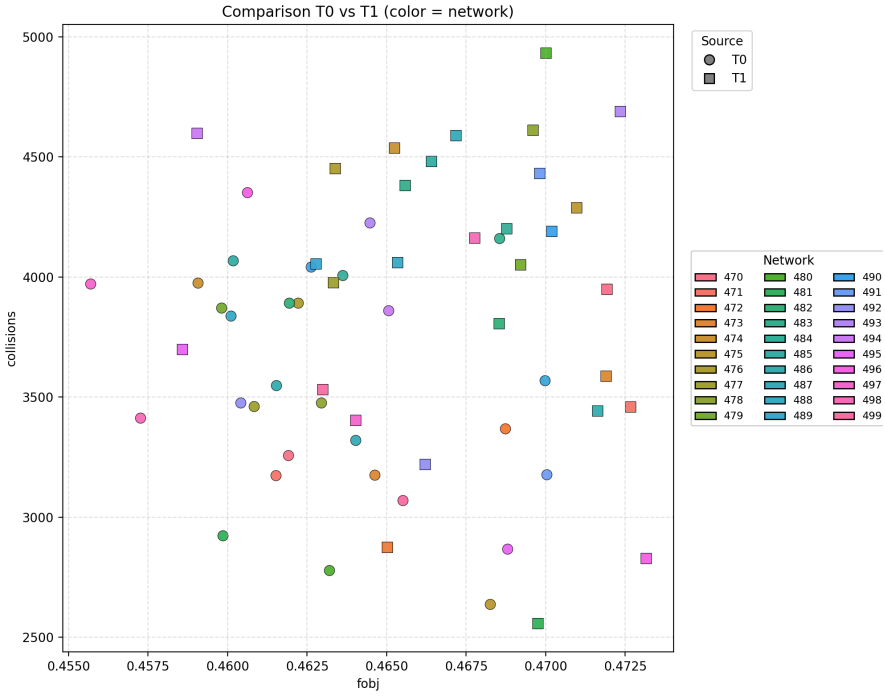
A notable limitation is the **elevated collision** count. We hypothesize that this arises from a discrepancy between offline and online evaluation conditions: networks were optimized and selected under a fixed initial footbot position during offline evolution, whereas online adaptation evaluates each epoch with varying starting locations. This positional variability may induce navigational difficulties, particularly given the architectural constraint of utilizing only two proximity sensors for arena traversal.

Damage analysis We introduced **sensor damage** at a magnitude `0.05`. Preliminary trials revealed that damage levels of `0.1` or higher induced persistent circular trajectories, as observed before, while `0` damage produced behavioral patterns qualitatively similar to those observed at `0.05`. The results obtained mirrored those observed prior to the implementation of offline evolution: the footbot exhibited improvement for average fitness values considering $T_0 - T_1$, Fig. fig. 3.4. As expected, networks subjected to sensor damage demonstrated reduced performance compared to their undamaged counterparts.

A significant asymmetry emerged depending on which sensor was compromised. When damage was applied to the **left** sensor, the system exhibited substantially higher collision counts but superior fitness values relative to **right** sensor damage. In both configurations, the footbot successfully navigated the arena perimeter; however, a pathological behavior manifested when left sensor damage coincided with clockwise arena traversal. Under these conditions, when the robot's left side contacted the perimeter wall while the left sensor was impaired, the footbot maintained continuous wall contact throughout the entire trajectory. This behavior persisted across epochs due to consistently elevated objective function values: the damaged **left** sensor continuously outputted `0.05`, while the **right** sensor registered `0` due to the absence of obstacles on that side, creating a stable, though dysfunctional, feedback configuration that the adaptation mechanism failed to correct.



(a)



(b)

Figure 3.4: Comparison $T_0 - T_1$ with damage `0.05`, applied to right fig. 3.4a and left fig. 3.4b sensors

Chapter 4

Multi-Sensor Analysis and Mechanism Comparison

4.1 Seven-Sensor Offline Simulation

Following the results obtained by analysing different scenarios with the 2-input and 2-output architecture, our objective was to increase the number of input values to enhance spatial resolution for obstacle detection on the ϵ-greedy decaying mechanism. Specifically, we choose to use a total of 14 proximity sensors, resulting in 7 input parameters, each calculated as the average of two adjacent physical sensors. These sensor pairs are arranged sequentially around the footbot's forward perimeter, as follows:

$\texttt{sensor\_pairs} = \{(6,5),(4,3),(2,1),(0,23),(22,21),(20,19),(18,17)\}$.

The objective was to resemble Floreano and Keller's setting, which employed 8 proximity sensors (6 forward-facing and 2 rear-facing) [FK10]. Due to the different disposition of the inputs, we decided to use 7 sensors all facing forward, so as not change too much the behavior of the footbot.

The expansion from two to seven inputs has required different changes to be done on the mechanism, starting with an increase in network parameters. For a hidden layer with H neurons, the parameter count is computed as in Equation eq. (2.2), the two-input number of parameters is 72, and for the seven-input variant with $H = 10$, $I = 7$, and $O = 2$ hidden neurons, we have:

$PARAMS = (7 \times 10) + 10 + 20 + (10 \times 2) + 2 = 122.$

The obstacle avoidance behavior is similar to before; it has only been adapted to the new sensor layout, and the obstacle avoidance behavior is called only when at least one proximity sensor value exceeds the proximity threshold of 0.1. If all sensors fall below this threshold, the robot executes random exploration, generating uncorrelated wheel velocities from a uniform distribution $(0, V_{max})$.

The obstacle avoidance or neural network logic for the seven-input architecture differs from its two-input counterpart, Algorithm algorithm 1. As shown in Algorithm algorithm 7, the hidden layer computation iterates over all seven sensor inputs for both cases, $H = 0$ and $H > 0$, and considers that all seven sensors influence the wheel velocities. At $H > 0$, the mathematical formulation for the hidden layer activation of neuron j is:

$$z_j = \sum_{i=0}^{6} w_{ij} \cdot s_i + b_j + w_{self} \cdot h_j^{prev} + w_{cross} \cdot h_{partner}^{prev} \tag{4.1}$$

where s_i represents the virtual sensor input value, w_{ij} are the input-to-hidden weights, b_j is the hidden bias, and w_{self}, w_{cross} are recurrent connection weights implementing temporal integration and lateral interaction between paired neurons.

Fitness computation also had to be adapted to the dimensions of the input sensors. We changed the `evaluate_performance()` function, which computes the instantaneous fitness, taking as the obstacle proximity component the maximum sensor reading across the 7 sensor array, as shown in Algorithm algorithm 8.

Algorithm 8 Fitness evaluation function - 7 sensors

1: **Function**: `evaluate_performance()`
2: $i \leftarrow \max($`array_sensors`$)$
3: `left_v_norm` $\leftarrow$ `normalize_velocity(left_v)`
4: `right_v_norm` $\leftarrow$ `normalize_velocity(right_v)`
5: $V \leftarrow ($`left_v_norm` $+$ `right_v_norm`$)/2$
6: **return** $V \cdot (1 - i)$

The enhanced spatial resolution provided by seven virtual sensors enabled more nuanced reactive behaviors, particularly in complex environments with multiple

Algorithm 7 `RNN_7`: Recurrent Neural Network with 7 inputs

function `RNN_7`(`array_sensors`)
  s $\leftarrow$ `array_sensors`
  if $H = 0$ **then**
    for $o \leftarrow 0$ **to** $O - 1$ **do**
      $z \leftarrow \sum_{i=1}^{I} w_{oi} \cdot s_i + b_o + \alpha_o \cdot$ `prev_output`$[o]$
      $out_o \leftarrow \text{sigmoid}(z)$
      `prev_output`$[o] \leftarrow out_o$
      if $o = 1$ **then**
        $v_L \leftarrow out_o \cdot v_{\max}$
      else
        $v_R \leftarrow out_o \cdot v_{\max}$
      end if
    end for
  else
    for $j \leftarrow 0$ **to** $H - 1$ **do**
      if j is odd **then**
        $\bar{j} \leftarrow j + 1$
      else
        $\bar{j} \leftarrow j - 1$
      end if
      $z \leftarrow \sum_{i=1}^{I} w_{ji} \cdot s_i + b_j + \alpha_{\texttt{self}} \cdot$ `prev_hidden`$[j] + \alpha_{\texttt{cross}} \cdot$ `prev_hidden`$[\bar{j}]$
      $h_j \leftarrow \text{sigmoid}(z)$
    end for
    for $o \leftarrow 0$ **to** $O - 1$ **do**
      $z \leftarrow \sum_{j=1}^{H} w_{oj} \cdot h_j + b_o$
      $out_o \leftarrow \text{sigmoid}(z)$
      `prev_output`$[o] \leftarrow out_o$
      if $o = 0$ **then**
        $v_L \leftarrow out_o \cdot v_{\max}$
      else
        $v_R \leftarrow out_o \cdot v_{\max}$
      end if
    end for
    `prev_hidden` $\leftarrow (h_1, h_2, \ldots, h_H)$
  end if
  return (v_L, v_R)
end function

nearby obstacles without the implementation of noise or damage, obtaining evolution fitness values in the offline simulation that achieved `0.49`.

4.1.1 Damage Analysis

Analogously to the two-sensor architecture, to assess the robustness of the controller, we selected and collected the best performing 30 configurations and conducted a series of experiments where we damaged proximity sensors and evaluated the mechanism's ability to recover and maintain control. To represent the sensor failure, we use a sensor override mechanism that replaces designated sensor indices with a fixed desired value (default: 0.05). The experiments were performed with online adaptation, and were performed as done previously, with malfunctions executed in groups. The parameters T_0, T_1, and T_{max} retain the same meaning established previously, referring to the fitness at the beginning, at the final epoch, and the best fitness reached during the run, respectively.

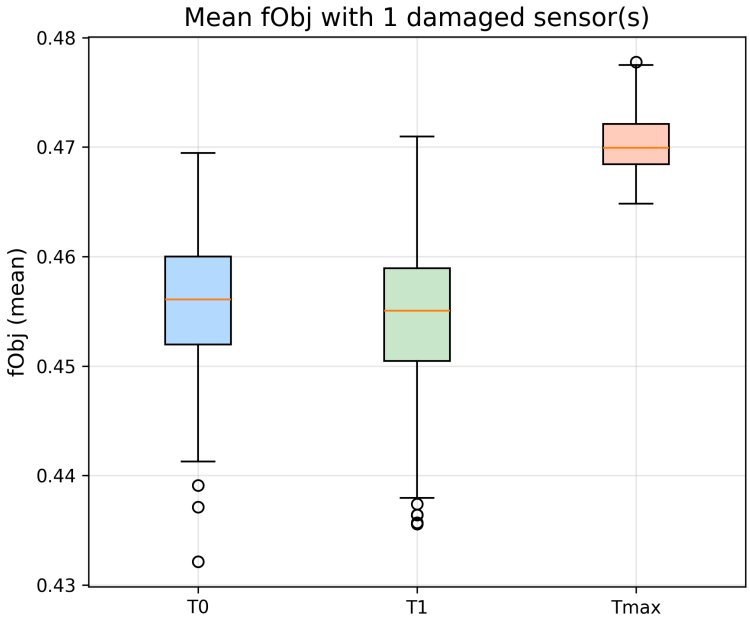


Figure 4.1: Comparison of T_0, T_1 and T_{max} `f_Obj` aggregated data with damage `0.05` applied to 1 sensor

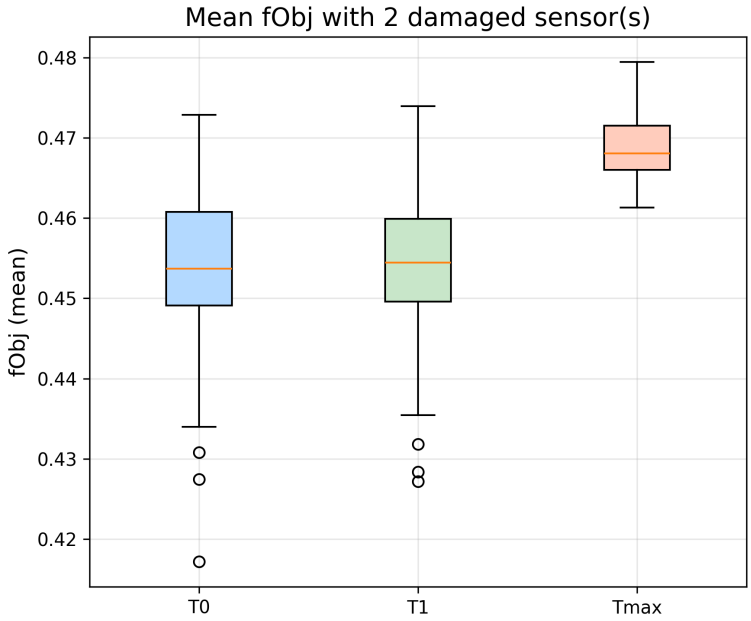


Figure 4.2: Comparison of T_0, T_1 and T_{max} `f_Obj` aggregated data with damage `0.05` applied to 2 sensors

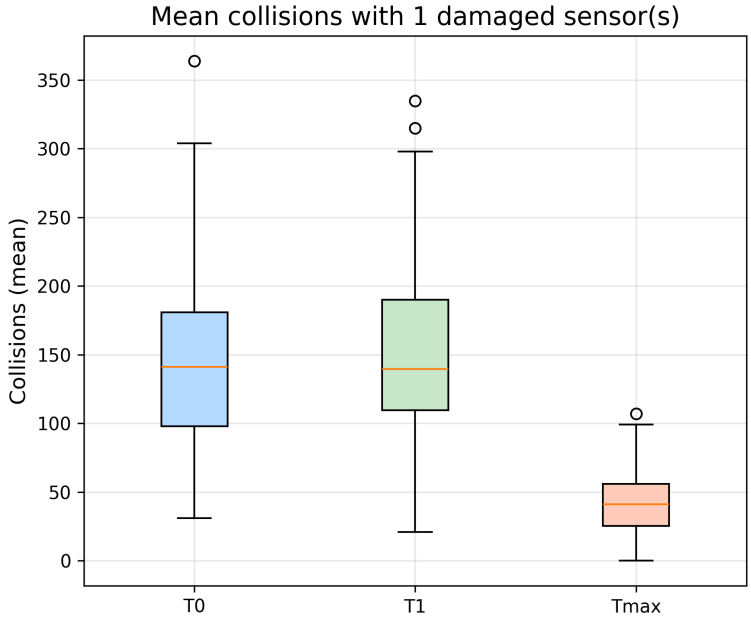


Figure 4.3: Comparison of T_0, T_1 and T_{max} `collisions` aggregated data with damage `0.05` applied to 1 sensors

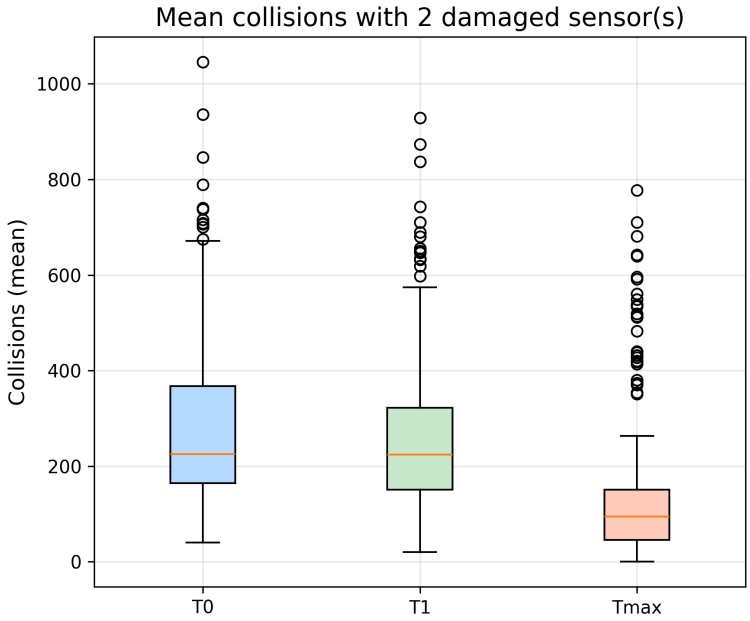


Figure 4.4: Comparison of T_0, T_1 and T_{max} `collisions` aggregated data with damage `0.05` applied to 2 sensors

One and Two sensors For these damage scenarios, **qualitative** behavioral analysis revealed minimal deviation from standard operation. The footbot maintained coherent obstacle avoidance trajectories and completed multiple circuits of the arena. However, **quantitative** fitness evaluation indicated a slight performance degradation with $f_{Obj} \sim 0.45$-0.46, compared to the undamaged case 0.48. For **one** damaged sensor, the average number of collisions is ~ 150, Fig. fig. 4.3; whereas for **two** sensors is around 270, Fig fig. 4.4.

Even with lower fitness values, the controller is able to successfully compensate for localized sensory malfunctions, maintaining a generally good behavior, apart from minor performance losses.

Three sensors With three damaged sensors, the footbot maintained a moderate f_{Obj} performance level, with the means being around $0.45 - 0.46$, albeit with a substantially elevated collision frequency of approximately 700–900 (see Fig. fig. 4.5 and Fig. fig. 4.6), with many outliers achieving numbers above 2000. The robot preserved the capability to follow the arena perimeter; however, the

frequency of obstacle contact increased greatly.

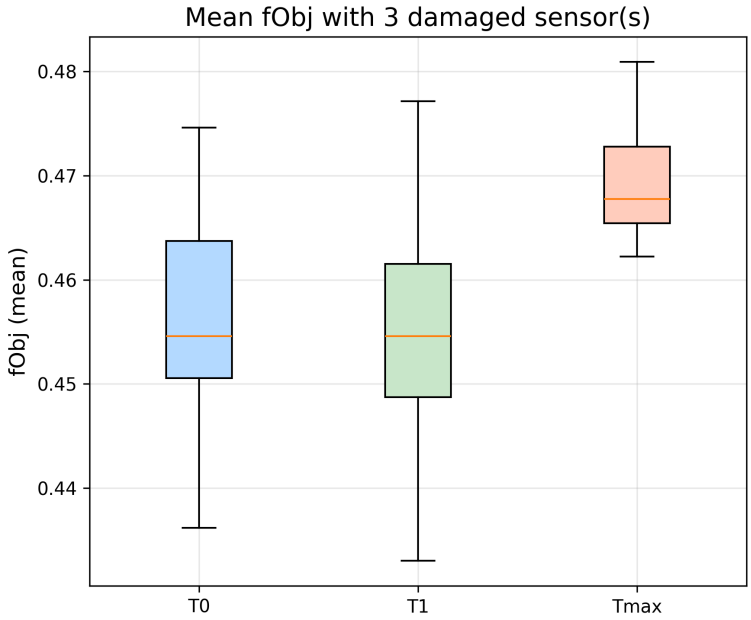


Figure 4.5: Comparison of T_0, T_1 and T_{max} `f_Obj` aggregated data with damage `0.05` applied to 3 sensors

Four sensors More pronounced degradation emerged with four or more disabled sensors. The footbot frequently became trapped in the arena, caused by the robot getting stuck against obstacles or arena walls. This happened predominantly with certain sensors being damaged, which aligned the location of the obstructing objects. As shown in Fig. fig. 4.7, this condition yielded a slightly higher fitness value of 0.46, but it is attributable to the controller maintaining relatively high velocities in the (incorrect) absence of perceived obstacles. In contrast, the actual number of collisions increased substantially, reaching approximately 2000 at T_0 and 2700 at T_1, observable in Fig. fig. 4.8. Despite this elevated collision rate, the controller was still occasionally able to complete full circuits of the arena, indicating a residual capacity for navigation under severely impaired sensory conditions.

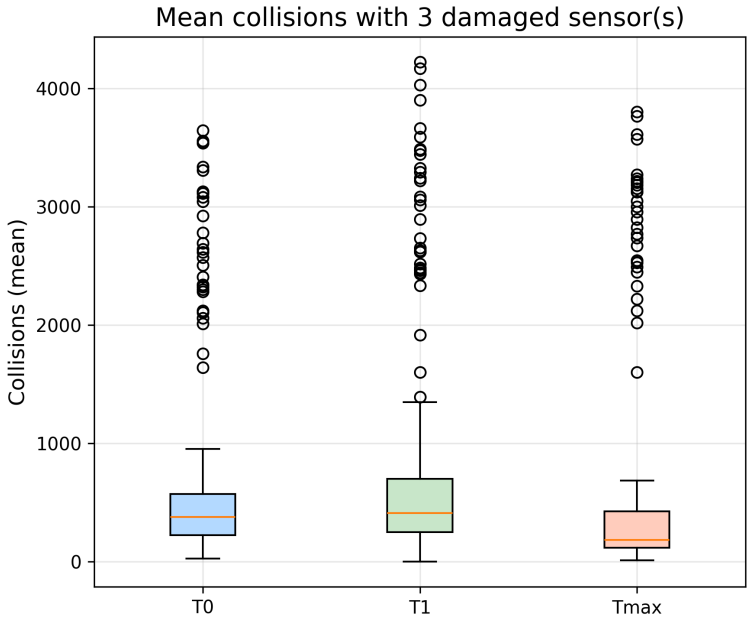


Figure 4.6: Comparison of T_0, T_1 and T_{max} `collisions` aggregated data with damage `0.05` applied to 3 sensors

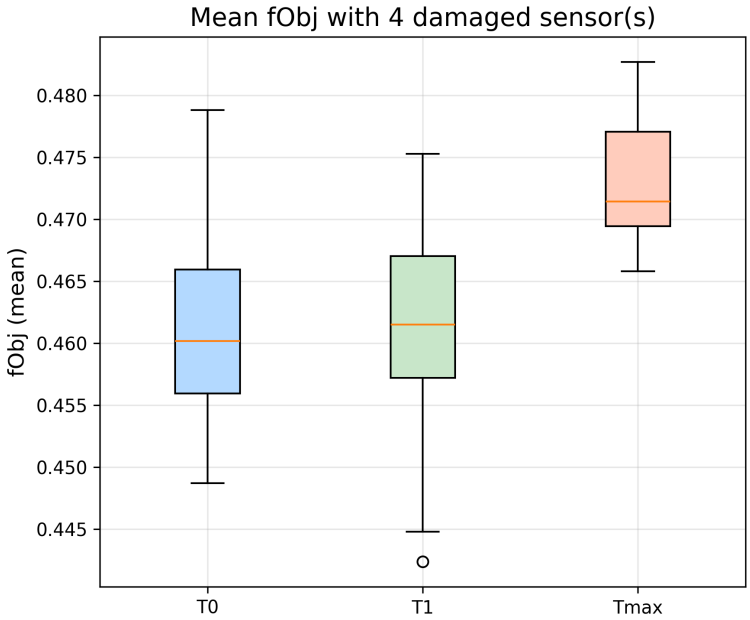


Figure 4.7: Comparison of T_0, T_1 and T_{max} `f_Obj` aggregated data with damage `0.05` applied to 4 sensors

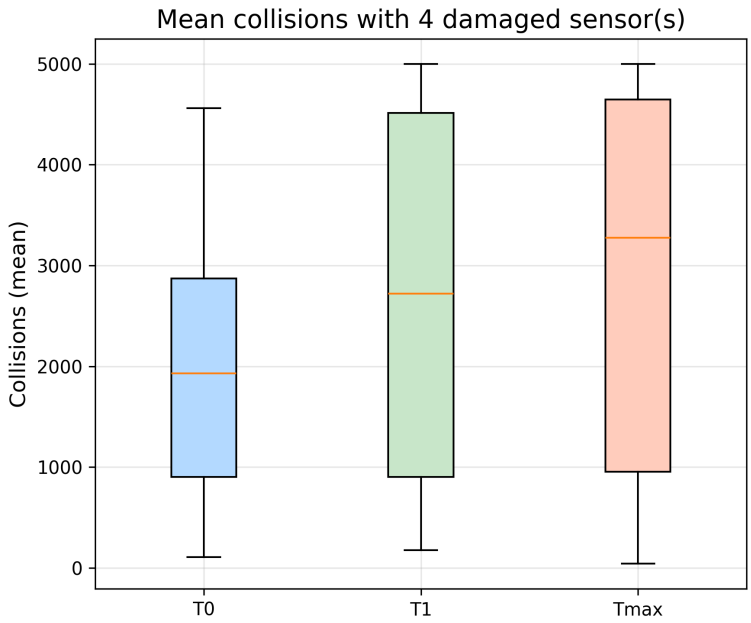


Figure 4.8: Comparison of T_0, T_1 and T_{max} `collisions` aggregated data with damage `0.05` applied to 4 sensors

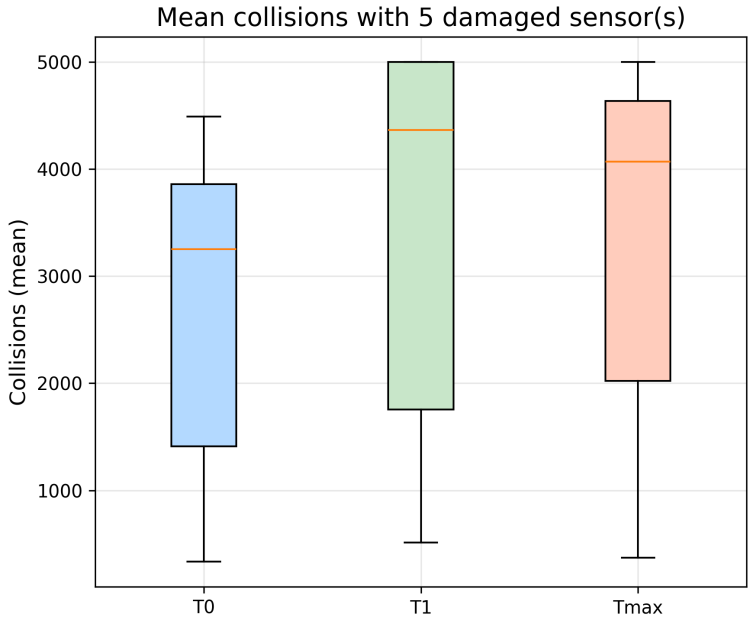


Figure 4.9: Comparison of T_0, T_1 and T_{max} `collisions` aggregated data with damage `0.05` applied to 5 sensors

Five sensors This layout produced very high collision frequencies (2700–3500) in most trials (Fig. fig. 4.9). Nonetheless, one or two outlier trials exhibited comparatively improved performance, achieving a fitness value of 0.465 (Fig. fig. 4.10). Even under these extreme conditions, the foot-bot retained a degree of exploratory behavior and sporadically completed full arena traversals, suggesting partial preservation of functional navigation capabilities despite profound sensory loss.

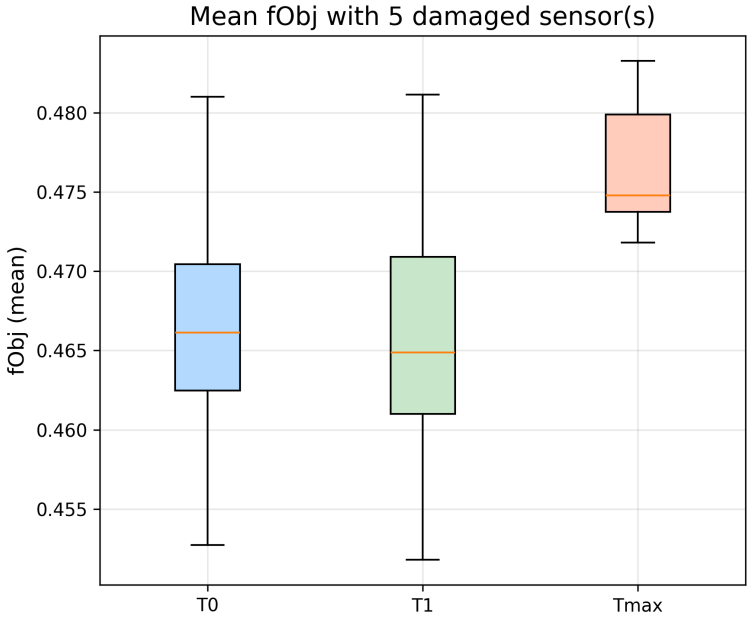


Figure 4.10: Comparison of T_0, T_1 and T_{max} `f_Obj` aggregated data with damage `0.05` applied to 5 sensors

Six sensors Failure of six sensors induced a qualitative shift in behavioral dynamics. The footbot displayed a marked tendency to become trapped in corners and to remain in prolonged contact with the arena perimeter, yielding collision counts on the order of 3000–4000 (Fig. fig. 4.11 and Fig. fig. 4.12).

A notable asymmetry in performance emerged as a function of which single sensor remained operational: configurations preserving the leftmost sensor supported substantially better mobility than those in which only the rightmost sensor was functional.

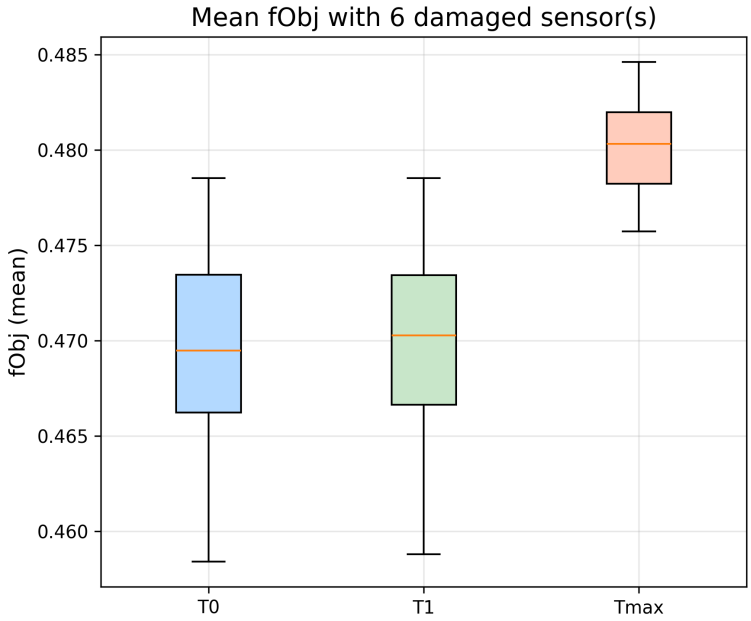


Figure 4.11: Comparison of T_0, T_1 and T_{max} `f_Obj` aggregated data with damage `0.05` applied to 6 sensors

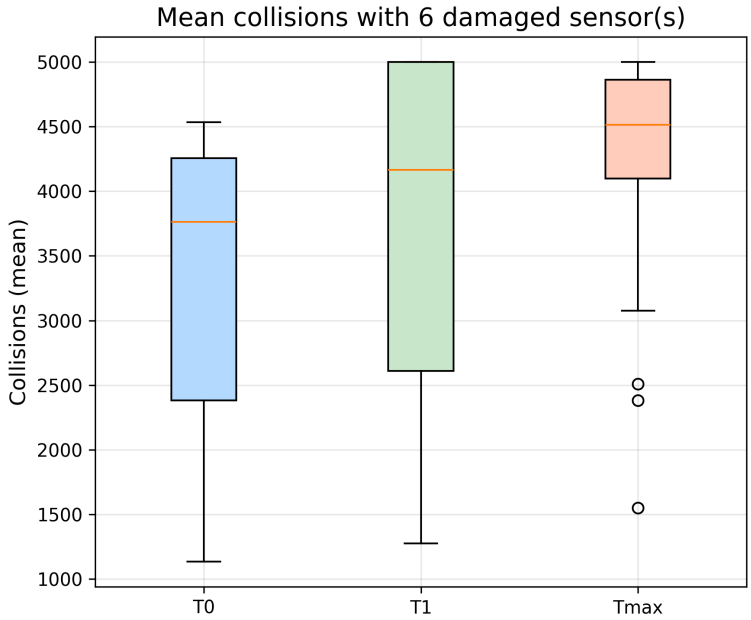


Figure 4.12: Comparison of T_0, T_1 and T_{max} `collisions` aggregated data with damage `0.05` applied to 6 sensors

Seven sensors With all seven sensors disabled, the robot experienced complete sensory failure, resulting in persistent collisions throughout the trials. From the initial epoch onward, the foot-bot remained trapped against arena structures, accumulating collision counts several orders of magnitude higher than baseline, in the range of 4000–5000 collisions per epoch (Fig. fig. 4.14 and Fig. fig. 4.13).

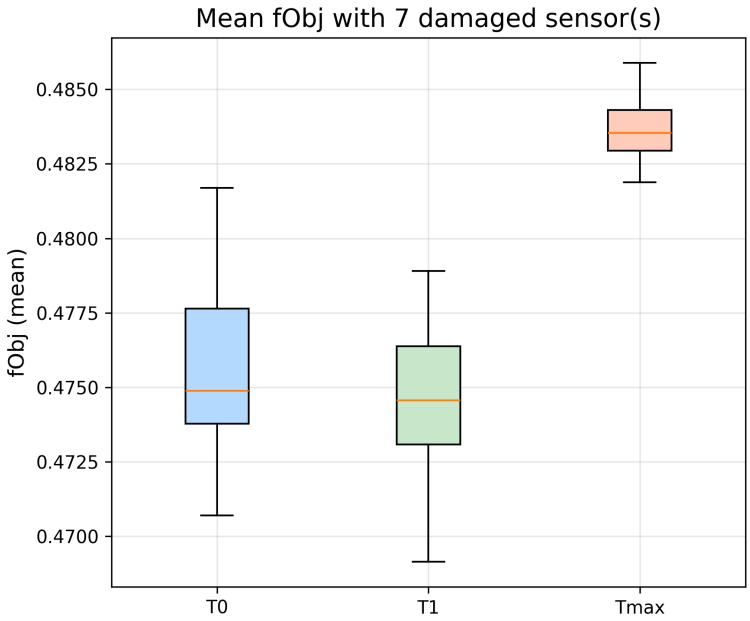


Figure 4.13: Comparison of T_0, T_1 and T_{max} `f_Obj` aggregated data with damage `0.05` applied to 7 sensors

The most notable observation from these experiments concerned the preservation of qualitatively normal behavior under light and mild damage (1–3 sensors). However, across all malfunction conditions, the results indicated only limited fitness improvement between trials: comparison of T_0 and T_1 revealed modest or even negative changes in f_{Obj}. This suggests that the mechanism is capable of maintaining a similar behavioral profile throughout the execution, but sensor damage leads to a slightly reduced performance level. The mechanism was unable to sustain peak fitness, as shown by the fact that $T_{\max}$ values consistently exceeded the corresponding T_1 measurements. This is likely due to changes in the spatial placement or orientation of the footbot between evaluations.

The asymmetry in controller configuration observed under 2-sensor damage

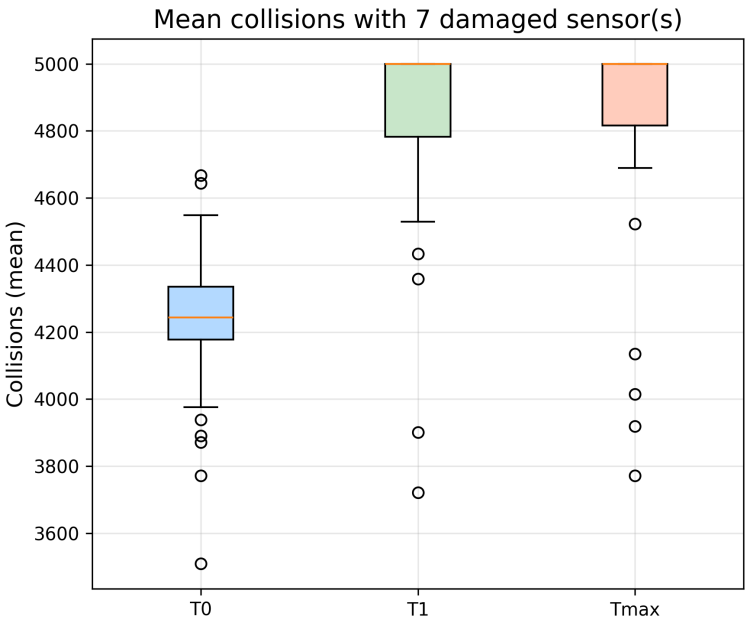


Figure 4.14: Comparison of T_0, T_1 and T_{max} `collisions` aggregated data with damage `0.05` applied to 7 sensors

conditions was also evident in the 7-input configuration: malfunctions of sensors on the left side resulted in markedly higher collision penalties than equivalent failures on the right side. This directional bias likely reflects the evolutionary dynamics encountered during offline training, where the fitness landscape favored right-turning strategies, presumably as a consequence of the initial robot placement and environmental layout in the arena.

4.2 Further adaptation mechanisms

After obtaining the results from the previous experiments, we decided to introduce new mechanisms that could be compared with ϵ-decaying, Algorithm eq. (2.6), and its behavior.

The *first* mechanism chosen was **Upper-Confidence-Bound** (**UCB**) action selection, which chooses actions in the multi-armed bandit setting that explicitly trade off exploitation and exploration. In this context, the two possible actions are exploration, consisting in generating a new configuration through mutation,

and exploitation, consisting in selecting the best known configuration We have chosen the most common equation of UCB, which is represented in *"Reinforcement Learning: An Introduction"* by Sutton and Barto [SB98], where actions are selected as:

$$A_t = \arg max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right], \tag{4.2}$$

where:

- $Q_t(a)$ is the estimated value (average reward) of action a up to time t.
- $c \in \mathbb{R}^+$ is an exploration constant controlling the degree of exploration.
- $N_t(a)$ is the number of times action a has been selected up to time t.
- t is the total number of decision steps up to the current time.
- $\ln(t)$ is the natural logarithm of t.

Rather than exploring randomly, the UCB approach aims to concentrate sampling on actions that either look promising or remain insufficiently sampled by augmenting each action's value with an exploration bonus, and then the action with the largest upper confidence bound is selected. [SB98] The implementation of the mechanism in our controller is described in Algorithm algorithm 9, in which, during the first 70 epochs, full exploration is executed, generating new configurations through random mutation without invoking the UCB rule. This initialization phase populates the action space with sufficient diversity to make the selection meaningful. Afterwards (epoch ≥ 70), the selection mechanism is activated, and as described in Algorithm algorithm 10, it is structured around two key phases: an initialization phase, followed by an exploration-exploitation phase. These phases are represented in the algorithm as:

- $N_a = 0$, actions that have never been tried.
- $N_a > 0$, actions that have been sampled at least once.

Algorithm 9 Online adaptation - UCB (Upper Confidence Bound)

Init: Q, $N \leftarrow$ arrays for value estimates and counts
$c \leftarrow$ exploration constant (1.0)
$\theta_i \leftarrow$ taken one of 30 network configurations
$Q[i] \leftarrow 0,\ N[i] \leftarrow 0$
for epoch **to** MAX_EPOCHS **do**
    `fitness_accum` $\leftarrow 0$
    for step **to** MOVE_STEPS **do**
        sense environment $\rightarrow$ `array_sensors`
        $(outL, outR) \leftarrow$ RNN_forward(`array_sensors`, θ_i)
        set motor speeds using outputs
        `fitness_accum` $\leftarrow$ `fitness_accum` + `evaluate_performance`()
    end for
    $f_{Obj} \leftarrow \frac{\texttt{fitness\_accum}}{\texttt{MOVE\_STEPS}}$
    $N[i] \leftarrow N[i] + 1$
    $Q[i] \leftarrow Q[i] + \frac{(f_{Obj} - Q[i])}{N[i]}$
    if epoch < 100 **then**
        exploration
    else
        idx $\leftarrow$ `select_action(Q, state, storage)`
        idx = index of next epoch config
    end if
end for

where N_a is the number of times action a has been selected. When untried actions remain in the pool, the algorithm performs *forced exploration*; this guarantees that every action eventually receives at least one evaluation before the scoring rule is applied. Once all actions have been sampled at least once, the algorithm enters the standard behavior, computing its values using Equation eq. (4.2).

Our implementation simplifies the problem as a *stateless global bandit* by maintaining a single state index (state = 0) throughout the algorithm's lifetime. The storage structure tracks two types of counters:

- `N(a)`: A map that stores $N_t(s, a)$ for each state-action pair. In our case, since there is only one state, this effectively tracks the number of times each configuration (action) has been selected.

- `T`: A map that stores the total visit count for each state, which corresponds

to the epoch counter t.

- `c`: The exploration constant, dynamically adjusted (normally 1.0).

Algorithm 10 UCB selection action

Function `ucb_action(Q, state, storage)`
$N(a) \leftarrow$ number of times action a has been selected
Phase 1: identify actions never evaluated
$\mathcal{U} \leftarrow \{a \mid N(a) = 0\}$
if $\mathcal{U} \neq \emptyset$ **then**
    $a^* \leftarrow$ uniform random choice from $\mathcal{U}$
else
Phase 2: all actions tried, compute UCB scores
    total selections $T = \sum_{a=1}^{K} N(a)$
    `best_score` $\leftarrow -\infty$
    for $a = 1$ **to** K **do**
        `bonus` $\leftarrow c \cdot \sqrt{\dfrac{\ln T}{N(a)}}$
        `score`$(a) \leftarrow \underbrace{Q(a)}_{\text{exploitation}} + \underbrace{\text{bonus}}_{\text{exploration}}$
        if `score`$(a) >$ `best_score` **then**
            `best_score` $\leftarrow$ `score`(a)
            $a^* \leftarrow a$
        end if
    end for
end if
$T \leftarrow T + 1$ ▷ increment total selection count
$N(a^*) \leftarrow N(a^*) + 1$ ▷ increment selected action count
return a^*

The *second* mechanism is the **Softmax/Boltzmann** algorithm, which employs probabilistic action selection via the Boltzmann distribution. The probability of selecting action a_i at state s is:

$$p(a_i|s) = \frac{e^{\frac{Q(s,a_i)}{\tau}}}{\sum_{j=1}^{|A|} e^{\frac{Q(s,a_j)}{\tau}}}, \tag{4.3}$$

where $|A|$ denotes the size of the action space, which consists of two actions: exploration and exploitation; $Q(s, a_i)$ is the estimated Q-value, and τ is the tem-

perature parameter. [WYCL23] This probabilistic framework balances exploration and exploitation through the temperature parameter τ, which controls the degree of exploration, as:

- τ is low, the distribution becomes *peaked* at the maximum Q-value, approaching greedy exploitation: $p(a_t) \to 1$ where $a = \arg\max_a Q(a_t)$.
- τ is high, the distribution becomes *uniform* over all actions, corresponding to uniform exploration: $p(a_t) \to \frac{1}{|A|}$.
- For intermediate values, the distribution smoothly interpolates between these extremes, assigning higher probabilities to high-performing actions while preserving non-zero probabilities for exploration.

The temperature value during the execution gradually shifts the algorithm from broad exploration to focused exploitation with:

$$\tau_{t+1} = \max(\tau_{\min}, \tau_t \times 0.995), \tag{4.4}$$

where $\tau_0 = 4.0$ and $\tau_{\min} = 0.1$. At each epoch, the Softmax selection routine is invoked, as defined in Algorithm algorithm 11.

Algorithm 11 Softmax/Boltzmann selection action

Function `softmax_action(Q, τ)`
$Q_{\max} \leftarrow \max_a Q[a]$
$K \leftarrow Q$ length
weights $\leftarrow \{\}$ $sum \leftarrow 0$
for $a = 0$ **to** $K - 1$ **do**
    numerically stable
    $weights[a] \leftarrow \exp\left(\frac{Q[a] - Q_{\max}}{\tau}\right)$
    $sum \leftarrow sum + weights[a]$
end for
for $a = 0$ **to** $K - 1$ **do**
    $p(a) \leftarrow \frac{w[a]}{\text{sum}}$
end for
return $idx \leftarrow$ discrete distribution over weights

In Algorithm algorithm 12, its described the implementation of softmax. As done previously, before configuration evaluation, the Q-values are updated via incremental averaging, Eq. eq. (2.5), and as well-performing configurations accumulate higher Q-values, their selection probabilities increase, naturally concentrating sampling on promising regions. In the implementation, like UCB, there is an initialization phase where, for 70 epochs, there is pure exploration. Afterwards, the Softmax mechanism is invoked, and the returned configuration index it provides is used for the next epoch. The mechanism selection logic is called as follows:

$$idx \sim \text{Softmax}(Q, \tau_t). \tag{4.5}$$

Algorithm 12 Online adaptation - Softmax (Boltzmann)

Init: Q, $N \leftarrow$ arrays for value estimates and counts
$\tau \leftarrow$ temperature parameter (default 2.0)
$\tau_{\min} \leftarrow 0.1$ ▷ temperature lower bound
$\theta_i \leftarrow$ taken one of 30 network configurations
$Q[i] \leftarrow 0,\ N[i] \leftarrow 0$
for epoch **to** MAX_EPOCHS **do**
    `fitness_accum` $\leftarrow 0$
    for step **to** MOVE_STEPS **do**
        sense environment $\rightarrow$ `array_sensors`
        $(outL, outR) \leftarrow$ `RNN_forward(array_sensors,`θ_i`)`
        set motor speeds using outputs
        `fitness_accum` $\leftarrow$ `fitness_accum` + evaluate_performance()
    end for
    $f_{Obj} \leftarrow \frac{\texttt{fitness\_accum}}{\texttt{MOVE\_STEPS}}$
    $N[i] \leftarrow N[i] + 1$
    $Q[i] \leftarrow Q[i] + \frac{(f_{Obj} - Q[i])}{N[i]}$
    if epoch < 70 **then**
        forced exploration
    else
        idx of next epoch $\leftarrow$ `softmax_action(`Q`,`τ_t`)`
        $\tau \leftarrow \max(0.1, \tau \times 0.995)$
    end if
end for

The *third* mechanism implemented was **VDBE-Boltzmann**, taken from *"Value-*

Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax" by Tokic and Palm. [TP11] Its idea is to extend the ϵ-greedy method by controlling a state-dependent exploration probability, $\epsilon(s)$, in dependence on the temporal difference (TD) error, instead of manual tuning. VDBE follows the exploration and exploitation logic, which calls for the agent to explore when there is a lack of environmental knowledge, as indicated by significant shifts in the value function. Then, as the agent increases its knowledge about the environment, and there are small or no changes in the value function, the exploration rate is reduced. The VDBE mechanism integrates into the controller through one primary interface, the ε adaptation, Algorithm algorithm 13. The different logic is placed at the conclusion of each epoch, after the assessment of the average epoch reward and updating Q-values, the algorithm:

1. Computes the TD-error for the selected configuration.
2. Applies the Boltzmann transform to obtain $f(s_t, a_t, \sigma)$.
3. Computes a dynamic smoothing factor $\delta = 1/|A|$.
4. Updates $\varepsilon_{t+1}(s)$ according to the exponential moving average formula.
5. Clamps $\varepsilon_{t+1}(s)$ to $[\varepsilon_{\min}, \varepsilon_{\max}]$.

In this way, the exploration rate is continually adjusted based on the most recent observations while maintaining stability through smoothing and bounds.

The temporal difference (TD) error is defined as:

$$\text{TD-error} = f_{Obj} - Q_t(a) \tag{4.6}$$

where f_{Obj} is the reward obtained in the current epoch and $Q_t(a)$ is the prior estimate of the configuration's value. This error quantifies the discrepancy between the observed reward and the algorithm's current belief about how good the selected configuration is.

Rather than directly using the TD-error magnitude to control exploration, VDBE applies a nonlinear Boltzmann transformation to map the error into a well-behaved exploration signal, shown in Eq. eq. (4.7):

Algorithm 13 Online adaptation - VDBE

Init:Q, $N \leftarrow$ arrays for value estimates and counts
$\sigma, \leftarrow 0.05,\ \epsilon \leftarrow 1.0$
$\theta_i \leftarrow$ take one of 30 network configurations
$Q[i] \leftarrow 0,\ N[i] \leftarrow 0$
for epoch **to** `MAX_EPOCHS` **do**
    `fitness_accum` $\leftarrow 0$
    for step **to** `MOVE_STEPS` **do**
        sense environment $\rightarrow$ `array_sensors`
        $(outL, outR) \leftarrow$ `RNN_forward`(`array_sensors`, θ_i)
        set motor speeds using outputs
        `fitness_accum` $\leftarrow$ `fitness_accum` + `evaluate_performance`()
    end for
    $f_{Obj} \leftarrow \frac{\texttt{fitness\_accum}}{\texttt{MOVE\_STEPS}}$
    $N[i] \leftarrow N[i] + 1$
    $\alpha \leftarrow \frac{1}{N[i]}$
    $\Delta Q \leftarrow (f_{Obj} - Q_{\text{old}})$
    $Q[i] \leftarrow Q[i] + \frac{\Delta Q}{N[i]}$

    $x \leftarrow -\frac{|\alpha \cdot \Delta Q|}{\sigma}$
    $f \leftarrow \frac{1-e^x}{1+e^x}$

    $|A| \leftarrow$ number of arms
    $\delta \leftarrow \frac{1}{|A|}$
    $\epsilon \leftarrow \delta \cdot f + (1 - \delta) \cdot \epsilon$
    $\epsilon \leftarrow \max(0.1, \min(0.9, \epsilon))$
    $p \leftarrow$ random uniform in $[0, 1]$
    if $p \leq \epsilon$ **then**
        exploration
        $\theta_{i+1} \leftarrow$ `mutate`$(\theta_i),\ Q[i+1] \leftarrow 0,\ N[i+1] \leftarrow 0$
    else
        exploitation
        $\theta_i \leftarrow \theta_j \quad$ where $j = \arg\max_k Q[k]$
    end if
end for

$$f(s,a,\sigma) = \frac{1 - e^{\frac{-|\alpha \cdot TD-Error|}{\sigma}}}{1 + e^{\frac{-|\alpha \cdot TD-Error|}{\sigma}}}, \tag{4.7}$$

σ is the sensitivity parameter controlling how sensitive the transformation is to changes in the TD-error, our ΔQ. The larger σ is the less responsive to TD-errors, while a smaller σ makes it more responsive. In our experiment, we tried different values for σ, and decided to choose 0.05

The exploration probability $\epsilon_t(s)$ is adapted at each epoch according to:

$$\epsilon_{t+1}(s) = \delta \cdot f(s_t, a_t, \sigma) + (1 - \delta) \cdot \epsilon_t(s) \tag{4.8}$$

where δ is a smoothing factor (typically $\delta \approx 1/|A|$, the reciprocal of the action space size). $|A|$is the action space size, where in this case, the two possible actions are exploration and exploitation. This adaptive choice ensures that as the algorithm progresses, the smoothing factor α naturally decreases as the selected configuration accumulates evaluations, making the epsilon adaptation increasingly conservative over time and gradually shifting the balance from exploration towards exploitation. Additionally, we enforce bounds on ϵ to prevent pathological behavior:

$$\epsilon_{t+1}(s) = \max(\epsilon_{\min}, \min(\epsilon_{\max}, \epsilon_{t+1}(s))) \tag{4.9}$$

in which $\varepsilon_{\min}$ and $\varepsilon_{\max}$ are user-specified lower and upper bounds ($\varepsilon_{\min} = 0.1$, $\varepsilon_{\max} = 0.9$). These bounds ensure that the algorithm never becomes purely exploitative nor purely exploratory.

4.3 Experimental Evaluation

Considering these new controllers, we tried them with the footbot using the 30 virtually evolved networks. The performance obtained ranged from 0.46 to 0.49, and the number of collisions went up to $150 - 200$, similarly to ϵ-greedy. Due to these results, we wanted to compare how Softmax, UCB, and VDBE would fare against ϵ-greedy with damaged sensors.

4.3.1 Robustness to Sensor Damage

To assess the effectiveness of three new mechanisms, we conducted the same series of experiments where we damaged proximity sensors and evaluated the algorithms' ability to recover and maintain control performance. Unlike in previous experiments, we added a new parameter to measure performance: T_e, a mean reference that encompasses data from epochs $1-5$. For each mechanism, all data of every scenario were documented in two types of tables: one for f_{Obj} and the other for *collisions*. The shown values were aggregated across multiple runs and reported as means computed as absolute measures at T_0,T_e, T_1, and $T_{\max}$. For ease of reading, the 7 setups are referred to as $s_1, s_2, s_3, s_4, s_5, s_6$ and s_7. Note that collision values reported in all tables are rounded to the nearest integer for ease of reading.

Softmax

Mechanism Softmax/Boltzmann - f_{Obj}				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	0.4561	0.4546	0.4545	0.4703
s_2	0.4548	0.4534	0.4536	0.4690
s_3	0.4550	0.4551	0.4549	0.4694
s_4	0.4620	0.4609	0.4614	0.4730
s_5	0.4659	0.4654	0.4646	0.4764
s_6	0.4702	0.4696	0.4694	0.4800
s_7	0.4743	0.4747	0.4761	0.4835

Table 4.1: Softmax/Boltzmann f_{Obj} data across all 7 scenarios with 0.05 damage for the four parameters: T_0,T_e, T_1, and $T_{\max}$

In Table table 4.1 and table 4.2 the data shows that at one sensor damage (s_1) the system shows limited immediate adaptation, with f_{Obj} declining from $T_0 = 0.456$ to $T_e = 0.455$ and remaining stable at T_1, suggesting minimal progressive improvement through the training window. However, T_{max} represents a net gain of +0.014 over initial performance, accompanied by a collision reduction from 149 at T_1 to 41 at T_{max}. As damage severity increases to three sensors (s_3), an interesting phenomenon emerges: $T_e = 0.455$ slightly exceeds $T_0 = 0.455$ in average reward, though collision frequency increases by 93 events. The peak performance

Mechanism Softmax/Boltzmann - *collisions*				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	135	147	149	41
s_2	272	287	282	141
s_3	845	938	952	750
s_4	1959	2504	2612	2650
s_5	2693	3342	3436	3339
s_6	3428	3671	3642	4288
s_7	4226	4812	4679	4853

Table 4.2: Softmax/Boltzmann collisions data across all 7 scenarios with 0.05 *damage for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

of $T_{max} = 0.469$ achieves collision levels (750) below the initial post-damage state, demonstrating effective recovery despite the intermediate degradation. The total damage scenario (seven sensors) reveals $T_1 = 0.476$ surpassing $T_0 = 0.474$, while collisions increase progressively ($4226 \rightarrow 4812 \rightarrow 4679 \rightarrow 4853$), confirming that the selected by Softmax systematically trades collision avoidance for higher velocity.

UCB

Mechanism UCB - f_{Obj}				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	0.4558	0.4542	0.4544	0.4703
s_2	0.4545	0.4537	0.4544	0.4693
s_3	0.4557	0.4554	0.4551	0.4696
s_4	0.4611	0.4609	0.4621	0.4727
s_5	0.4663	0.4652	0.4654	0.4762
s_6	0.4697	0.4699	0.4701	0.4799
s_7	0.4757	0.4751	0.4739	0.4835

Table 4.3: UCB f_{Obj} *data across all 7 scenarios with* 0.05 *damage for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

For s_1, Table table 4.3 and Table table 4.4, UCB shows a mild initial degradation from T_0 to T_e, with T_1 remaining stable at 0.454 before reaching T_{max} with a net gain of +0.014 accompanied by a sharp collision drop to just 41. With two

Mechanism UCB - *collisions*				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	139	150	150	41
s_2	275	284	281	149
s_3	822	941	1028	727
s_4	2085	2616	2706	2788
s_5	2958	3312	3386	3332
s_6	3228	3694	3679	4243
s_7	4304	4832	4817	4845

Table 4.4: UCB collisions data across all 7 scenarios with 0.05 *damage for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

sensors damaged, it achieves the highest T_{max} among all mechanisms, halving collisions from 281 to 149; notably, the initial decline from T_0 to T_e is almost entirely recovered by T_1, demonstrating effective intermediate adaptation. A distinctive pattern emerges in the tables at four sensors, where T_1 actually exceeds T_0, indicating that the exploration manages to push performance beyond the original baseline during the adaptation process; however, collisions increase monotonically through all time points, revealing a growing disconnect between fitness optimization and collision control. This divergence becomes most apparent at six sensors, where T_1 again surpasses T_0 and peak performance reaches $T_{max} = 0.480$, yet collisions at T_{max} (4243) far exceed those at T_1 (3679), confirming that the best-performing theta configuration sacrifices collision avoidance to maximize average reward.

VDBE

VDBE exhibits the strongest initial robustness among all mechanisms, observable with the data in Table table 4.5 and Table table 4.6 for s_1, the $T_{max} = 0.471$ yields both the highest average reward and the lowest collision count (39) across all mechanisms at this level. At two sensors, a distinctive pattern appears: $T_1 = 0.454$ falls below T_e in both reward and collision control (290 vs 281 collisions), indicating that prolonged adaptation can temporarily deteriorate performance relative to the early phase, before $T_{max} = 0.469$ compensates with halved collisions. At three sensors, VDBE achieves the highest peak among all mechanisms ($T_{max} = 0.470$),

Mechanism VDBE - f_{Obj}				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	0.4561	0.4545	0.4549	0.4705
s_2	0.4550	0.4540	0.4535	0.4692
s_3	0.4561	0.4548	0.4553	0.4699
s_4	0.4614	0.4610	0.4619	0.4729
s_5	0.4664	0.4655	0.4653	0.4764
s_6	0.4699	0.4697	0.4687	0.4800
s_7	0.4752	0.4747	0.4756	0.4831

Table 4.5: VDBE f_{Obj} data across all 7 scenarios with 0.05 damage for the four parameters: T_0,T_e, T_1, and $T_{\max}$

Mechanism VDBE - *collisions*				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	135	147	148	39
s_2	280	281	290	148
s_3	805	947	951	744
s_4	2010	2460	2615	2804
s_5	2784	3291	3316	3352
s_6	3248	3757	3636	4393
s_7	4188	4866	4778	4808

Table 4.6: VDBE collisions data across all 7 scenarios with 0.05 damage for the four parameters: T_0,T_e, T_1, and $T_{\max}$

bringing collisions (744) below the initial T_0 level (805). For moderate damage (s_4), $T_1 = 0.462$ surpasses $T_0 = 0.461$, and VDBE records the lowest intermediate collision count at T_e, indicating superior early adaptation management. At six sensors, collisions at T_{max} reach 4393, the highest among all mechanisms, despite $T_{max} = 0.480$ representing the strongest reward recovery from T_e.

Comparative Analysis

Across all four mechanisms, f_{Obj} exhibits a counter-intuitive trend: it increases with the number of damaged sensors, probably due to the fitness function $f_{Obj} = V \cdot (1 - i)$ not being able to withstand damaged sensors that return low proximity readings, artificially inflating the fitness regardless of actual obstacle proximity. As a consequence, the adaptation process across all mechanisms converges toward

high-velocity configurations that appear optimal under the distorted sensory input, but are in fact collision-prone.

The four mechanisms produce remarkably similar f_{Obj} trajectories. Due to this fact, we did not do a thorough analysis of the fitness values, mainly because we wanted to find which mechanism produced the least amount of collisions. To comprehensively assess each mechanism against one another with epsilon-greedy, we compared their mean collision results at absolute measures (T_1 and T_{max}) and relative recovery measures (T_1/T_e and T_{max}/T_e).

Collisions - Final epoch At T_1, for s_1, VDBE achieves the lowest collision count (147), while the decaying mechanism performs worst (150), as observable in Figure fig. 4.15. At s_2, this relationship inverts: decaying yields the best performance (272 collisions) while VDBE exhibits the highest (290). In s_3, decaying maintains superiority (917 collisions) with UCB performing worst (1028). Lastly, s_4+, VDBE and Softmax emerge as the most effective collision controllers.

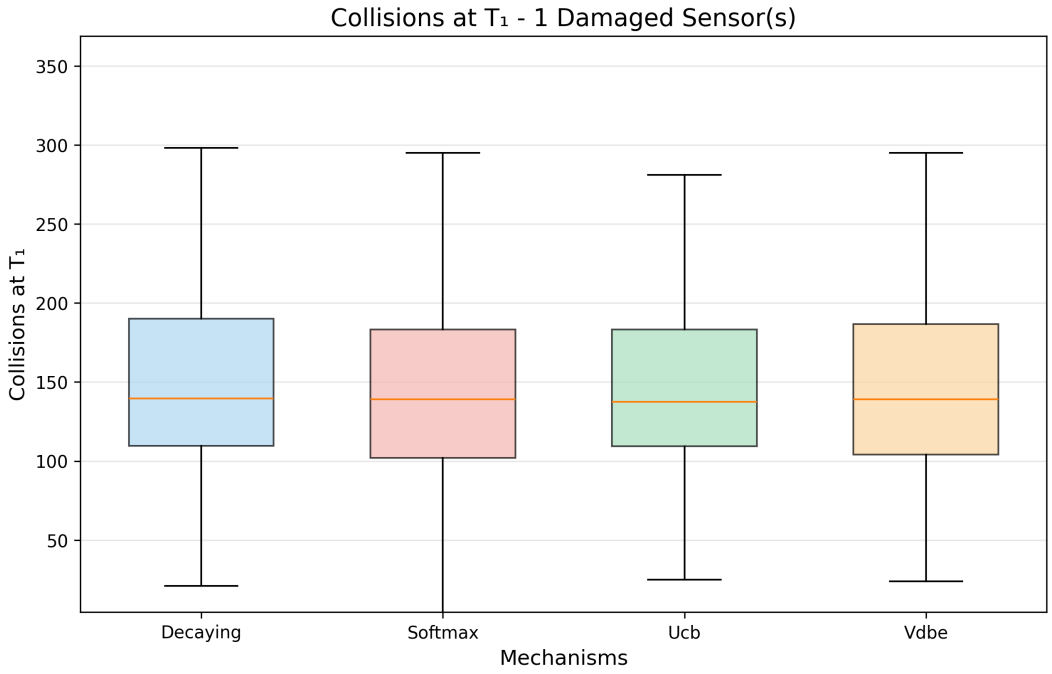


Figure 4.15: Comparison of 4 mechanisms at T_1 of `collisions` aggregated data with damage `0.05` applied to 1 sensor

Collisions - Best epoch T_{max}, show different dynamics. At s_1, Fig. fig. 4.16, VDBE achieves the minimum collision count (38), though the margin over other mechanisms is narrow. Softmax has the best results with 140 collisions at s_2 with decaying being the worst (150). For s_3, decaying performs best (722) while Softmax yields the highest collisions (750). Critically, at s_4+, T_{max} collision levels exceed those at T_1, indicating that the best-performing policy configuration prioritizes average reward optimization over collision minimization at severe damage levels.

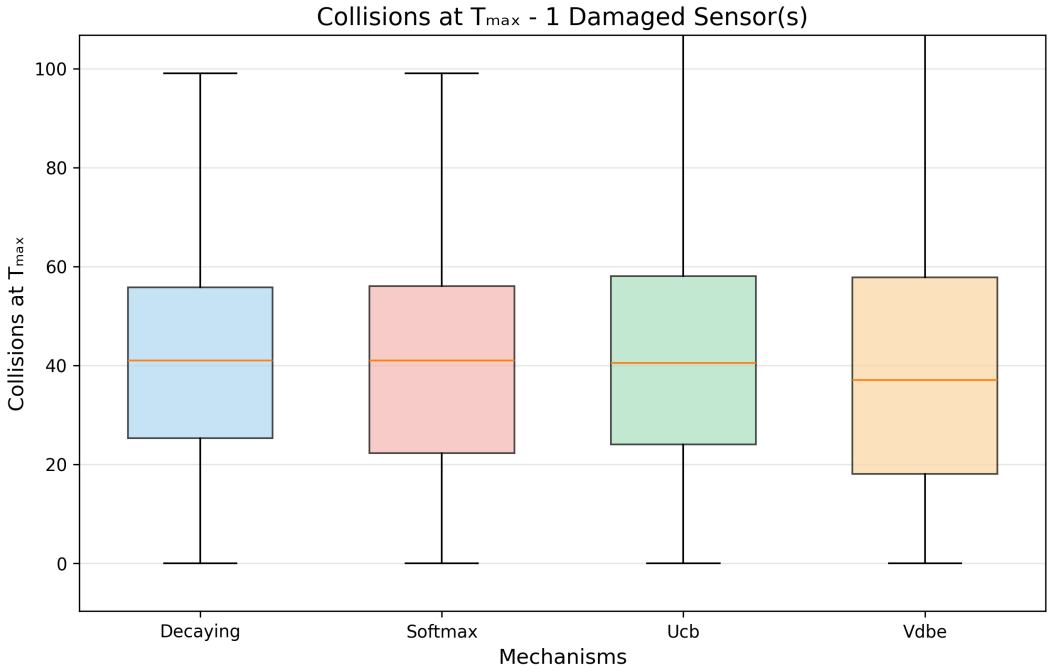


Figure 4.16: Comparison of 4 mechanisms at T_{max} of `collisions` aggregated data with damage `0.05` applied to 1 sensor

Collisions - Recovery ratio T_1/T_e This ratio quantifies improvement from early degradation to the final epoch. With s_1, all mechanisms achieved sporadic $4-5\%$ collision reduction to T_e, however the events were engulfed by the rest of the data were the predominance was balance between T_1 and T_e values, with UCB showing the best ratio of 1.03. At s_2, decaying demonstrates the highest improvement with 0.99 (Figure fig. 4.17). However, at s_3+, recovery is minimal

with ratios all surpassing 1.0.

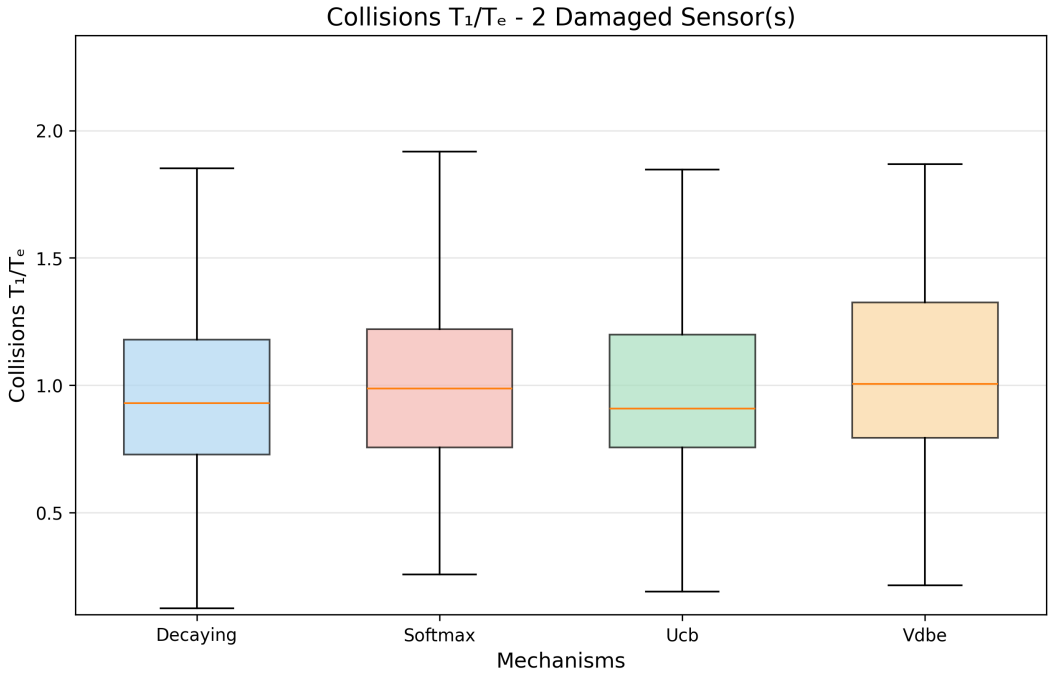


Figure 4.17: Comparison of 4 mechanisms at T_1/T_e of `collisions` aggregated data with damage `0.05` applied to 2 sensor

Collisions - Recovery ratios T_{max}/T_e Peak recovery capacity can help in revealing more substantial improvements. For s_1, UCB and VDBE achieve dramatic collision reductions of up to $\sim 70-75\%$ relative to T_e, with values respectively of 0.26 and 0.27, shown in Figure fig. 4.18. At s_2, Softmax maintains strong performance with $\sim 60\%$ reduction with a ratio of 0.41. With s_3, UCB demonstrates the best recovery capacity at 0.56. However, at s_4+, we do not have any improvements; we notice instead that T_{max} and T_e have similar data. The analysis reveals that no single mechanism demonstrates clear superiority across all conditions. For objective function performance, final differences among mechanisms remain negligible (less than 1%), indicating functional equivalence in reward optimization. Collision performance shows a larger variation, with VDBE preferred for single sensor damage and decaying mechanisms for moderate damage levels. At peak performance, collision control remains broadly similar under light damage, with

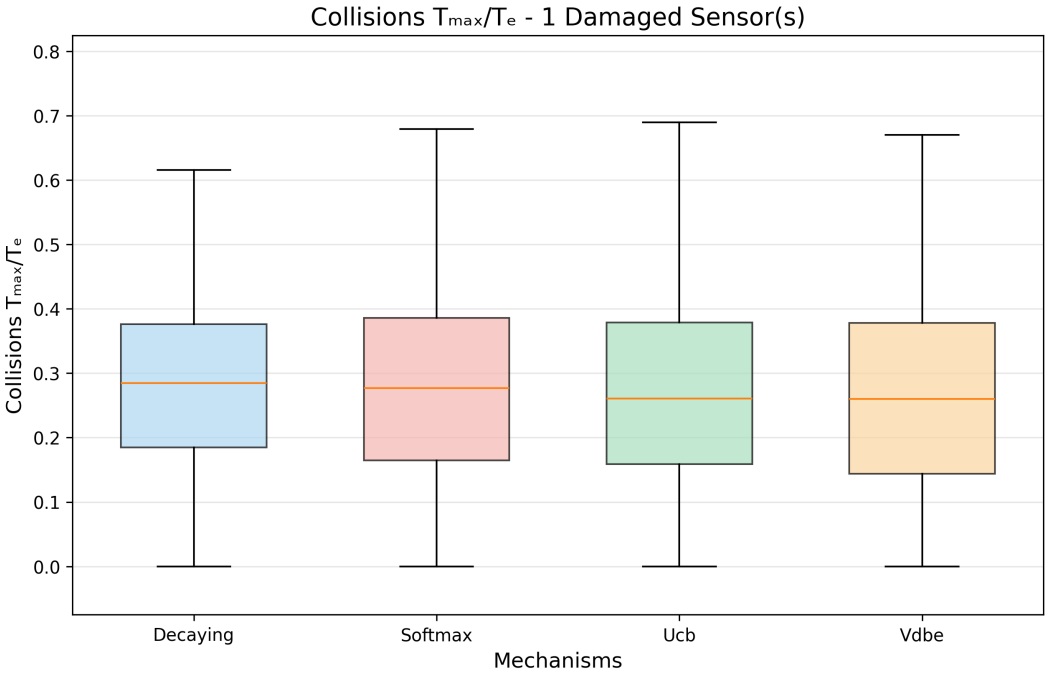


Figure 4.18: Comparison of 4 mechanisms at T_{max}/T_e of `collisions` aggregated data with damage `0.05` applied to 1 sensor

decaying showing marginal advantages at moderate severity.

The most salient finding across all mechanisms is the recovery ratio between T_{max} and T_1 results. This gap, observable in both reward and collision metrics, indicates that the optimal policy discovered during adaptation frequently occurs well before the final epoch, suggesting that current training protocols may benefit from earlier stopping criteria or more sophisticated convergence detection.

Chapter 5

Robustness to Penalty

The 7-sensor architecture provided more insightful results compared to the 2-input counterpart; however, we still noticed discrepancy between the fitness values and the number of collisions. For this reason, we thought a solution could be to increase the influence collisions have on the performance by adding a more incisive penalty, early stopping, and exclusively using the best configuration after a number of epochs.

The new `evaluate_performance()` function, shown in Algorithm algorithm 14, includes this new penalty, where in case of collision the fitness value is decreased by 0.5. Subsequently, we run our four mechanisms to observe if this addition produced different results compared to previous instances. We ran the executions with a damage of 0.05 to the sensors with the same strategy as before.

Algorithm 14 Fitness evaluation function - Penalty

1: $i \leftarrow \max(\texttt{array\_sensors})$
2: $\texttt{left\_v\_norm} \leftarrow \texttt{normalize\_velocity}(\texttt{left\_v})$
3: $\texttt{right\_v\_norm} \leftarrow \texttt{normalize\_velocity}(\texttt{right\_v})$
4: $V \leftarrow (\texttt{left\_v\_norm} + \texttt{right\_v\_norm})/2$
5: $\texttt{raw\_fitness} \leftarrow V \cdot (1 - i)$
6: $\texttt{collision\_penalty} \leftarrow \begin{cases} 0.5 & \text{if collision detected} \\ 0.0 & \text{else} \end{cases}$
7: **return** $\max(0, \texttt{raw\_fitness} - \texttt{collision\_penalty})$

Regarding the early-stopping process instead described in Algorithm algo-

Algorithm 15 Online Adaptation with Early Stopping

Init:Q, $N \leftarrow$ arrays for value estimates and counts
$c \leftarrow 0.5,\ \epsilon \leftarrow 1.0,\ \tau \leftarrow 4.0$
$\theta_i \leftarrow$ take one of 30 network configurations
$Q[i] \leftarrow 0,\ N[i] \leftarrow 0$
$f_{best} \leftarrow 0,\ Q_{best} \leftarrow 0$
`counter` $\leftarrow 0$
for epoch **to** `MAX_EPOCHS` **do**
    `fitness_accum` $\leftarrow 0$
    for step **to** `MOVE_STEPS` **do**
        sense environment $\rightarrow$ `array_sensors`
        $(outL, outR) \leftarrow$ `RNN_forward`(`array_sensors`, θ_i)
        set motor speeds using outputs
        `fitness_accum` $\leftarrow$ `fitness_accum` + `evaluate_performance`()
    end for
    $f_{Obj} \leftarrow \frac{\texttt{fitness\_accum}}{\texttt{MOVE\_STEPS}}$
    $N[i] \leftarrow N[i] + 1$
    $Q[i] \leftarrow Q[i] + \frac{(f_{Obj} - Q[i])}{N[i]}$
    ▷ *Early stopping logic*
    if epoch < 100 **then**
        if $f_{epoch} > f_{best}$ **then**
            $f_{best} \leftarrow f_{epoch},\ Q_{best} \leftarrow Q[i]$
        end if
    else
        if $f_{epoch} > f_{best}$ **then**
            $f_{best} \leftarrow f_{epoch},\ Q_{best} \leftarrow Q[i]$
            `counter` $\leftarrow 0$
        else
            `counter` $\leftarrow$ `counter` $+ 1$
            if `counter` ≥ 150 **then** ▷ early-stopping activated
                $\tau, \epsilon, c \leftarrow 0.01$
            end if
        end if
    end if
end for

rithm 15, it can be distinguished in two cases:

- Before epoch 100: we only save the best epoch obtained.

- After epoch 100: we begin a counter that has to reach 150, the system's patience, to enable early stopping.

Subsequently to 100 epochs, at the end of each epoch we check whenever the current fitness is greater than the registered best one, f_{best} and if it is, then we save that f_{Obj} as new best, we save the index of the current Q and we reset the counter. When the counting arrives at the patience value, the early-stopping is activated, forcing the mechanism to only exploit the best configuration by lowering ϵ, c, and τ to 0.01.

The mechanisms were executed with these new logic implemented, using the same manner of logic for damaged sensors as before. The data was collected as mean values in tables divided by f_{Obj} and *collisions* for each scenario, where collision values reported in all tables are rounded to the nearest integer for ease of reading. Throughout this chapter, the notation T_0, T_1, T_e, and T_{max} follows the conventions established in the previous chapters.

Softmax

Mechanism Softmax/Boltzmann - f_{Obj}				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	0.4574	0.4548	0.4543	0.4709
s_2	0.4467	0.4444	0.4446	0.4656
s_3	0.4040	0.3939	0.3922	0.4411
s_4	0.3156	0.2703	0.2419	0.3955
s_5	0.2505	0.2075	0.1888	0.3464
s_6	0.2205	0.1761	0.1816	0.3053
s_7	0.1390	0.0797	0.0860	0.1989

Table 5.1: Softmax/Boltzmann f_{Obj} data across all 7 scenarios with 0.05 *damage and penalty for the four parameters: T_0,T_e, T_1, and $T_{\max}$*

In Table table 5.1 the f_{Obj} data of T_{max} exceeds T_0 at every damage level, meaning the adaptation always finds a better solution than the initial damaged configuration. The gap between T_1 and T_e grows noticeably at s_4, s_6, and s_7, suggesting a temporary performance dip during the mid-adaptation phase that is fully resolved at the end.

Mechanism Softmax/Boltzmann - *collisions*				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	114	133	121	32
s_2	244	268	249	95
s_3	785	903	888	385
s_4	1889	2436	2677	928
s_5	2703	3227	3323	1536
s_6	3097	3639	3574	2059
s_7	4121	4848	4772	3383

Table 5.2: Softmax/Boltzmann collisions data across all 7 scenarios with 0.05 *damage and penalty for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

Softmax achieves the lowest absolute collision count at mild damage: s_1 drops to just 32 at T_{max}, the best value recorded across all mechanisms. At high damage levels, observable in Table table 5.2 its collision reduction at T_{max} remains competitive, though not the strongest.

UCB

Mechanism UCB - f_{Obj}				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	0.4568	0.4547	0.4545	0.4711
s_2	0.4469	0.4449	0.4435	0.4654
s_3	0.4035	0.3922	0.3868	0.4415
s_4	0.3119	0.2699	0.2555	0.3959
s_5	0.2488	0.2046	0.1977	0.3416
s_6	0.2090	0.1736	0.1726	0.3088
s_7	0.1363	0.0805	0.0822	0.1955

Table 5.3: UCB f_{Obj} *data across all 7 scenarios with* 0.05 *damage and penalty for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

Regarding its f_{Obj} results, shown in Table table 5.3, UCB is the only mechanism where T_1 never exceeded T_e at any damage level, the gap remained small and consistently negative. This reflects its execution, which distributes trials evenly without sharp oscillations.

In Table table 5.4, the number of *collisions* of UCB achieved low T_{max} at several mid-range damage levels like s_3, and s_6. However, at extreme damage (s_7),

Mechanism UCB - *collisions*				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	119	135	120	30
s_2	246	262	253	93
s_3	795	925	960	384
s_4	1935	2440	2519	919
s_5	2734	3260	3221	1594
s_6	3233	3669	3680	2012
s_7	4155	4839	4817	3431

Table 5.4: UCB collisions data across all 7 scenarios with 0.05 *damage and penalty for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

it retained the highest residual collisions, confirming that its cautious exploration yields less collision reduction when the damage is most severe.

VDBE

Mechanism VDBE - f_{Obj}				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	0.4573	0.4541	0.4542	0.4711
s_2	0.4476	0.4446	0.4444	0.4655
s_3	0.4047	0.3883	0.3904	0.4413
s_4	0.3141	0.2592	0.2573	0.3965
s_5	0.2494	0.2009	0.1935	0.3440
s_6	0.2117	0.1734	0.1759	0.3052
s_7	0.1359	0.0807	0.0754	0.2019

Table 5.5: VDBE f_{Obj} *data across all 7 scenarios with* 0.05 *damage and penalty for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

VDBE delivers the best f_{Obj} recovery at s_7 with $T_{max} = 0.2019$ (Table table 5.7). Its exploration rate adapts dynamically to environmental changes, making it the most effective when a large number of sensors are lost. The cost is the deepest mid-adaptation valley: $T_1 = 0.0754$ at s_7, the lowest value recorded across all mechanisms and time points.

The number of *collisions* with this mechanism exhibits the most extreme swing: at s_7, collisions peak at 4903 (T_1), yet drop to 3356 at T_{max}. It also achieves the very low value for T_{max} collisions at s_1 (29) as observable in Table table 5.6.

Mechanism VDBE - *collisions*				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	115	140	125	29
s_2	244	265	252	94
s_3	779	971	916	385
s_4	1905	2569	2505	916
s_5	2720	3304	3269	1564
s_6	3201	3671	3642	2052
s_7	4161	4834	4903	3356

Table 5.6: VDBE collisions data across all 7 scenarios with 0.05 *damage and penalty for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

Decaying Epsilon-Greedy

Mechanism ε-decaying - f_{Obj}				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	0.4566	0.4550	0.4539	0.4707
s_2	0.4475	0.4446	0.4438	0.4659
s_3	0.4012	0.3901	0.3882	0.4410
s_4	0.3035	0.2708	0.2607	0.3961
s_5	0.2472	0.2017	0.2007	0.3480
s_6	0.2128	0.1720	0.1806	0.3050
s_7	0.1389	0.0783	0.0788	0.1993

Table 5.7: ϵ*-decaying* f_{Obj} *data across all 7 scenarios with* 0.05 *damage and penalty for the four parameters:* T_0,T_e, T_1, *and* $T_{\max}$

The adaptation effect in this layout with ϵ-greedy is minor for s_1-s_2, where all time points remain within a narrow band ($\sim 0.44 - 0.47$). T_1 surpasses T_e only from s_6 onward, suggesting that the adaptation enables earlier partial recovery. All the data is reported in Table table 5.7.

In Table table 5.8, the results of collision with the ϵ-decaying mechanism give us the possibility to understand its progress across all the scenarios. In particular, we can see ϵ-decaying showing the highest baseline collision count at mild damage, hinting at a less efficient starting behaviour. However, T_{max} at s_5 is very good, confirming its strength in mid-range convergence.

Mechanism ε-decaying - *collisions*				
N. dmg sensors	T_0	T_e	T_1	T_{max}
s_1	125	130	124	31
s_2	242	265	252	90
s_3	816	950	939	389
s_4	2039	2429	2463	924
s_5	2748	3296	3177	1515
s_6	3189	3688	3584	2056
s_7	4118	4864	4854	3382

Table 5.8: ϵ-decaying collisions data across all 7 scenarios with 0.05 *damage and penalty for the four parameters: T_0,T_e, T_1, and $T_{\max}$*

5.1 Comparative Analysis

Regarding f_{Obj} from the results obtained from the four different mechanisms, we saw that the adaptation becomes more effective as the damage severity increases. We also noticed:

- with mild damage (s_1, s_2), performance remained stable across T_0, T_e, and T_1, with only a slight degradation, with T_{max} providing a modest improvement.
- with severe damage (s_4-s_7), a significant performance drop is observed from T_0 to T_1, but usually in s_6 and s_7, T_1 values increased compared to T_e, showing that the footbot improved its behavior at the end of the execution, related to its beginning.

For *collisions*, the data gave us the possibility to understand the mechanisms' progress across all the scenarios. In particular, we could saw that:

- collisions increased drastically with the number of damaged sensors, ranging from $\sim 110 - 130$ in s_1 to ~ 4500 in s_7.
- T_1 tended to exhibit fewer collisions than T_e, indicating that the damage did not worsen the robot's behavior. This, however, was not always observable, see at s_4.
- T_{max} consistently reduced collisions compared to the other parameters, in some cases with even a drop of $\sim 70\%$, like in s_1.

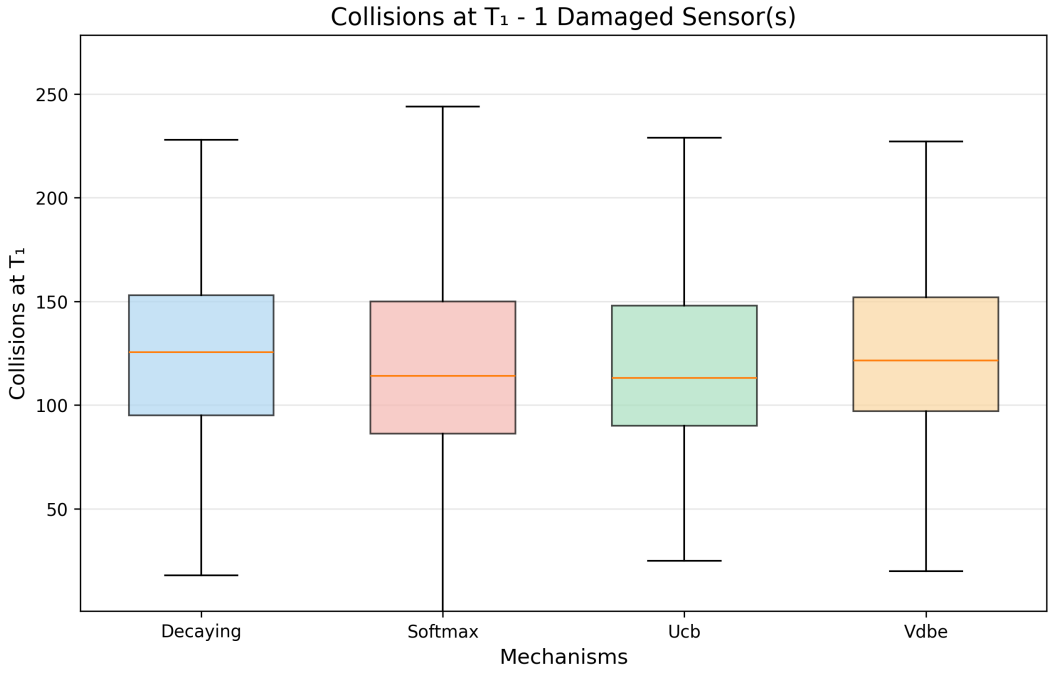


Figure 5.1: Comparison of 4 mechanisms at T_1 of `collisions` aggregated data with damage `0.05` applied to 1 sensor

Collisions - Final epoch At T_1, for s_1 (Fig. fig. 5.1), UCB achieved the lowest collision count (119), while VDBE is the worst (125). At s_2, softmax yielded the best performance (248 collisions) while decaying, and UCB exhibited the highest (252). In s_3, Softmax maintained superiority (888 collisions) with UCB performing worst (959). Decaying emerged as the mechanism with the best values at s_4 (2463), and s_5 (3177). Softmax has the lowest ratio at s_6 (3574) and s_7 (4771).

Collisions - Best epoch T_{max}, showed different dynamics. At s_1, VDBE obtained the minimum collision count (29) as observable in Figure fig. 5.2. Decaying had the best results with 90 collisions at s_2, with Softmax being the worst (95). For s_3, UCB and VDBE performed best (384) while Decaying yielded the highest collisions (389). In s_4, VDBE has the lowest number of collisions (915), Decaying for s_5 (1515), UCB at s_6 (2011), and in s_7, VDBE with 3356.

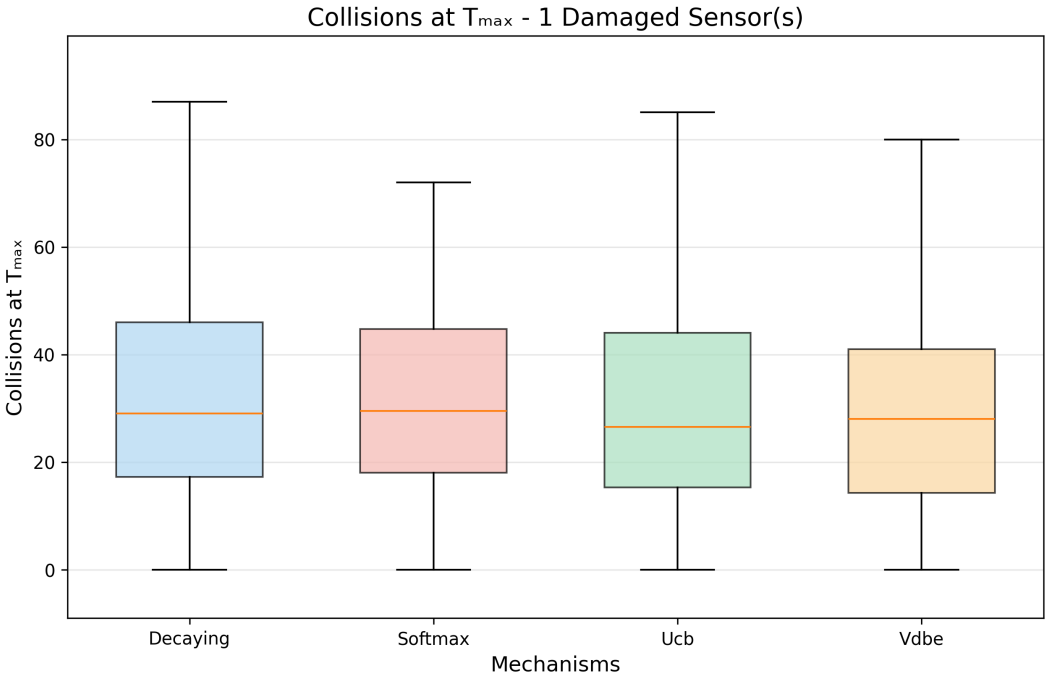


Figure 5.2: Comparison of 4 mechanisms at T_{max} of `collisions` aggregated data with damage `0.05` applied to 1 sensor

Collisions - Recovery ratio T_1/T_e With s_1, UCB attained the lowest recovery ratio of 0.91, with instead Decaying having the highest at 0.98, Fig. fig. 5.3. At s_2, Softmax demonstrates the greatest improvement with 0.93. However, at s_3 and s_4, recovery was not present with ratios all surpassing by a little 1.0. Notable is the fact that, at s_5, s_6, and s_7, there was a recovery with the best ratios obtained with: Decaying for s_5 and s_6 (0.96), and lastly UCB (0.99) for s_7.

Collisions - Recovery ratios T_{max}/T_e For s_1, VDBE and UCB achieved respectively of 0.21 and 0.22 with a reduction of up to $\sim 75-80\%$ (Fig. fig. 5.2. At s_2, Softmax, together with UCB, maintained strong performance with $\sim 70\%$ reduction with a ratio of 0.329. With s_3, VDBE scored the best recovery at 0.29. The ratios are very similar between the four mechanisms in s_4 (~ 0.30), without one predominating over the others. At s_5, Decaying has the lowest ratio with 0.37. With s_6, UCB and VDBE have both 0.47, and lastly at s_7, we have the four

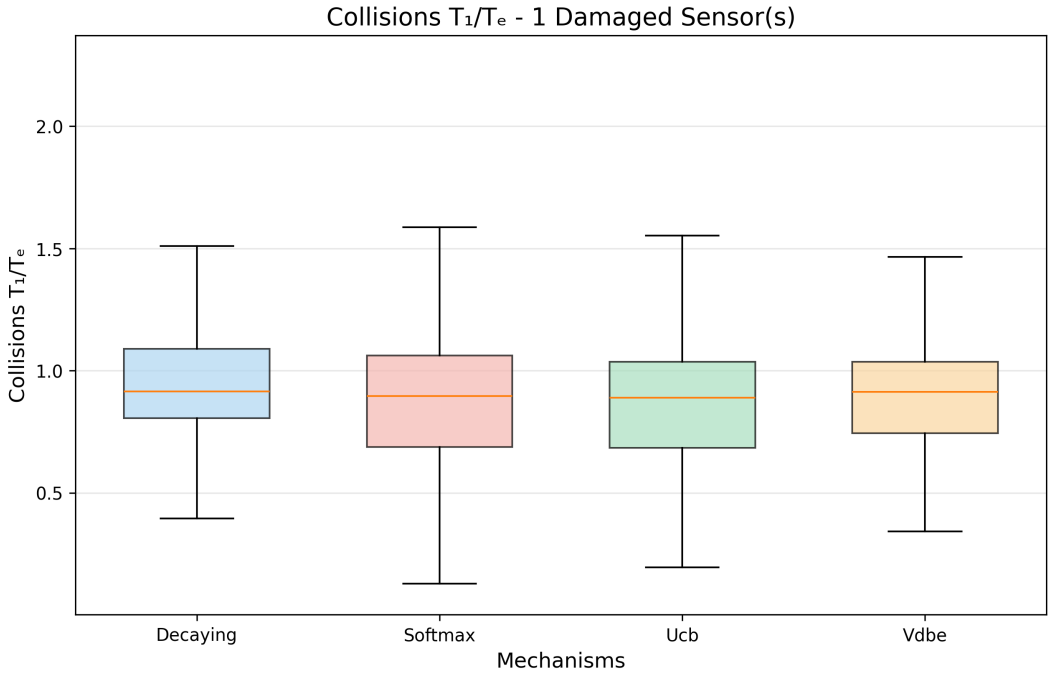


Figure 5.3: Comparison of 4 mechanisms at T_1/T_e of `collisions` aggregated data with damage `0.05` applied to 1 sensor

mechanisms balancing around $0.69 - 0.70$, reaching a recovery of 30%.

5.1.1 Penalty versus Recovery Framework Comparison

The introduction of a penalty produced interesting improvements in collision counts across all damage levels and mechanisms. To better grasp the picture, we directly compared the penalty-based data with the previously obtained results, named recovery, across the four mechanisms and the seven damage scenarios.

Single Sensor Damage - s_1 At minimal damage, penalty implementation achieved strong collision reductions. For T_1, UCB demonstrated the lowest collision count (119.6) with 20.03% improvement over recovery, similar to the other mechanisms. At peak performance (T_{max}), VDBE achieved minimum collisions (29.1) with 25.18% improvement. Recovery ratios showed effectiveness: UCB attained $T_1/T_e = 0.91$ (9% improvement over early degradation), while VDBE

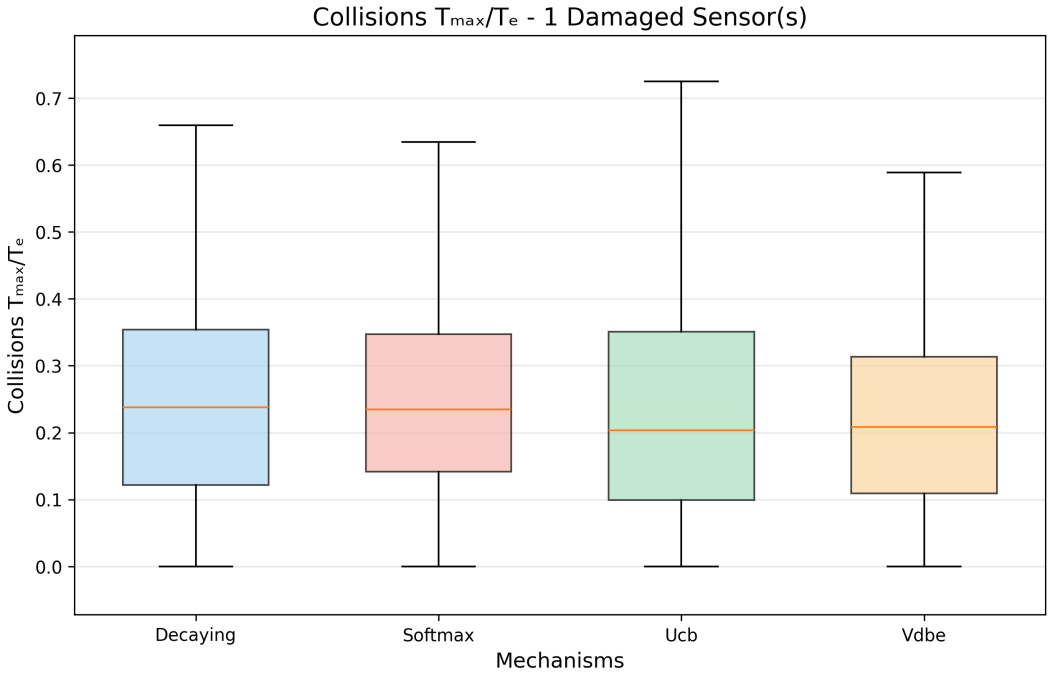


Figure 5.4: Comparison of 4 mechanisms at T_{max}/T_e of `collisions` aggregated data with damage `0.05` applied to 1 sensor

reaches $T_{max}/T_e = 0.21$ (79% reduction from baseline).

Two Sensors Damaged - s_2 With two sensors compromised, Softmax emerged strongest at T_1 with 248.7 collisions (11.70% improvement versus recovery). Decaying demonstrated superior performance with 90.7 collisions at T_{max} (39.74% improvement), being the largest percentage gain at this damage level. Recovery ratios confirmed efficacy with Softmax achieving $T_1/T_e = 0.94$ (6% improvement), while Decaying reached $T_{max}/T_e = 0.34$ (66% reduction).

Three Sensors Damaged - s_3 At moderate damage, Softmax maintained T_1 leadership with 888.3 collisions (6.70% improvement). UCB achieved the best performance with 384.3 collisions (47.12% improvement over recovery), the strongest T_{max} gain at this level. Notably, Softmax's $T_1/T_e = 1.01$ showed slight degradation (1% increase over the early phase), while UCB's $T_{max}/T_e = 0.42$ demonstrates robust 58% reduction.

Four Sensors Damaged - s_4 Moderate-high damage revealed Decaying's emergence: best T_1 performance with 2463.4 collisions (10.62% improvement). VDBE dominated peak metrics with 915.8 collisions (67.34% improvement over recovery). However, Decaying's $T_1/T_e = 1.01$ showed minimal degradation, while VDBE's $T_{max}/T_e = 0.36$ achieved 64% reduction from baseline.

Five Sensors Damaged - s_5 At serious damage, Decaying achieved comprehensive dominance: lowest T_1 collisions (3178.0, 9.19% improvement) and best T_{max} performance (1515.2 collisions, 55.22% improvement). Recovery ratios confirm superiority: $T_1/T_e = 0.97$ (3% improvement over early phase) and $T_{max}/T_e = 0.37$ (63% reduction), representing the most balanced performance across both metrics.

Six Sensors Damaged - s_6 Very high damage shows Softmax achieving minimum T_1 collisions (3574.4), though with a marginal 1.87% improvement over recovery. UCB demonstrated strong peak performance with 2011.8 collisions (52.58% improvement). Softmax maintained $T_1/T_e = 0.98$ (2% improvement), while UCB $T_{max}/T_e = 0.47$ (53% reduction from baseline).

Seven Sensors Damaged - s_7 Under catastrophic damage, Softmax showed 4771.5 collisions at T_1 but with negative improvement (-1.98% versus recovery), indicating penalty framework limitations at extreme damage. VDBE achieved the best peak with 3356.4 collisions (30.19% improvement). Softmax maintained $T_1/T_e = 0.98$, while VDBE's $T_{max}/T_e = 0.69$ represented a 31% improvement.

Several patterns emerge from this analysis comparison. Peak performance improvements (T_{max}) consistently exceed final epoch improvements (T_1) across all mechanisms and damage levels, with percentage gains ranging from 25% (s_1) to 67% (s_4), while T_1 improvements range from 2% (s_6) to 20% (s_1). The effectiveness of the penalty diminished with the increasing of failure severity:

- $s_1 - s_2$ improvements reached $11 - 40\%$ for both T_1 and T_{max}.
- $s_3 - s_4$ T_1 gained decline to $7 - 11\%$ while T_{max} maintained $47 - 67\%$.
- $s_5 - s_7$ T_1 improvements fell to $2 - 9\%$, while T_{max} dropped to $30 - 55\%$.

Recovery ratios reveal bimodal adaptation outcomes. At damage levels s_1, s_2, s_5, and s_6, T_1/T_e many ratios remained below or near 1.0, indicating successful collision reduction or stabilization from the early degradation phase. Conversely, at s_3 and s_4, ratios equal or exceed 1.0, meaning a slight degradation of T_1, and what we thought was that these moderate failure levels represent a critical boundary for the footbot, because we probably reach the limit of faulty sensors with which we can achieve still adequate qualitative behaviors. In fact, from the tables, we can notice that f_{Obj} at these levels begins decreasing more than the standard values collected in these settings, 0.45, reaching $0.40 - 0.30$. Over those levels, with s_5, s_6, and s_7, the robot does not really avoid obstacles since big portions of its sensors are malfunctioning; however, against this hypothesis, we have the recovery ratio T_{max}/T_e that showed improvement with values being substantially below 1.0, with a range of $0.21 - 0.69$.

A thing that remained was the asymmetry noticed in prior experiments. As we wrote about before, the data showed a big difference in the range of values when sensors on either the left side or right side were damaged. Subsequently, the behavior of the footbot changed greatly in either of these two situations, in particular, when the damage was on the left side of the robot, the collisions increased considerably, causing an abnormal demeanor compared to the values obtained in the other case.

In conclusion, the penalty framework altered the results of the exploration-exploitation mechanisms: discovering improved T_{max} and T_1 configurations, and conserved the inability to consistently maintain T_{max} configurations at T_1. This suggests that penalties may induce greater policy volatility, which can appear beneficial for discovering optimal solutions, and, with mild malfunctions, can achieve policy consolidation. However, it cannot be observed at moderate damage levels where the fitness landscape becomes most complex.

Chapter 6

Thymio Implementation

To assess whether the findings of this thesis extend beyond the simulated environment, the decaying ϵ-greedy seven sensors mechanism with penalty and early stopping was deployed on a physical Thymio robot. The controller configuration, θ, was selected directly from the repertoire of networks obtained through offline evolution.

Under nominal conditions, the Thymio exhibited coherent obstacle avoidance behavior, confirming that the evolved initialization repertoire transfers effectively to the physical platform. A video recording of this behavior is provided as supplementary material taken at epoch 5 (Video 1). To evaluate fault tolerance, a single proximity sensor was artificially damaged by fixing its reading to a constant value below the proximity threshold, replicating the damage model adopted throughout this thesis. Depending on the chosen position of the faulty sensor, following an initial degradation in movement, the adaptation mechanism successfully overcame the damage, recovering qualitatively adequate obstacle avoidance behavior. This outcome is consistent with the single-sensor damage results reported in previous chapters, where this scenario yielded the most favorable recovery dynamics across all mechanisms. Two video recordings document this recovery process: the first captures the robot's behavior at the beginning of the run in epoch 5 (Video 2), illustrating the initial performance degradation induced by the damage, while the second is recorded further along of the adaptation process at epoch 250 (Video 3), demonstrating the recovered obstacle avoidance capability. Additionally, a two-

sensor damage scenario was also tested on the physical platform. However, unlike the single-sensor case, no comparable recovery was observed, which is consistent with the greater difficulty of this scenario reported in the simulation.

Conclusion and Future Challenges

This thesis investigated the capacity of reinforcement learning mechanisms to maintain obstacle avoidance performance in a footbot robot subjected to progressive proximity sensor degradation. By combining offline evolution with online adaptation, we sought to find both the strengths and the structural limitations of four exploration-exploitation strategies across a comprehensive range of fault scenarios. By starting with selected strong networks that produced high results, we experimented with adding both noise and damages observing the produced results. Networks trained with only online adaptation gave us slightly lower results as opposed to the ones evolved offline. However, by starting with already well-trained networks, we observed that the optimal results did not align with the final epoch, indicating that the system was able to maintain normal behavior, comparable to that of the beginning epoch, with slight degradation, but showing performance deterioration measured up to the achieved peak. With the implementation of an added penalty on the offline networks, all four mechanisms achieved a lower number of collisions with a similar fitness across all damage conditions, revealing an improvement between the beginning and the end of the experiments, and compared to results that did not include this logic. In all cases, we were able to observe that with a low number of damaged sensors, the system is capable of maintaining its functionality, unlike when we have more than 3 sensors malfunctioning, which impairs its normal behavior. The application of one of the mechanisms in the real-world using Thymio as a subject shows that it is feasible to use adaptive behaviors from simulation to physical hardware. Future work could involve introducing noise or damage during a normal execution of the system in specific or random time steps, observing the system's ability to react to an unexpected malfunction providing a more realistic assessment of recovery dynamics and

adaptation stability. Additionally, it would be interesting to explore alternative adaptive strategies, such as Bayesian bandits or meta-learning approaches, that may improve stability.

In writing this dissertation, generative artificial intelligence tools (Visual studio GitHub Copilot, version 0.37 released in February 2026 and Claude Sonnet version 4.5 released in September 2025) were used for support in content creation and to improve the clarity of the text.

Bibliography

[BZL06] Josh Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[CCTM15] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nat.*, 521(7553):503–507, 2015.

[FK10] Dario Floreano and Laurent Keller. Evolution of adaptive behaviour in robots by means of darwinian selection. *PLoS biology*, 8(1):e1000292, 2010.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[Pac25] Benedetta Pacilli. Mechanisms for online adaptation in robots: A comparative study. 2025.

[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 Intell.*, 6(4):271–295, 2012.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[Ste85] Mark Stefik. V. braitenberg, vehicles: Experiments in synthetic psychology. *Artif. Intell.*, 27(2):246–248, 1985.

[TP11] Michel Tokic and Günther Palm. Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In Joscha Bach and Stefan Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings*, volume 7006 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2011.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Mach. Learn.*, 8:279–292, 1992.

[WYCL23] Xianjia Wang, Zhipeng Yang, Guici Chen, and Yanli Liu. A reinforcement learning method of solving markov decision processes: An adaptive exploration model based on temporal difference error. *Electronics*, 12(19), 2023.