Autonomous Control of Thymio Robots: Classical and Modern Approaches

Leonor Ramos (128936)[1], Tiago Patrício (105291)[1], João Almeida (127766)[1], David Mendes (25552)[1]

ISCTE – Instituto Universitário de Lisboa, Portugal

Abstract

This project explores the use of genetic algorithms, neural networks, and Deep Reinforcement Learning algorithms — namely PPO and Recurrent PPO with LSTM (Long Short-Term Memory) — for the development of an autonomous controller for the *Thymio* robot. The main objective of the project is to enable the *Thymio* robot to efficiently explore both simulated and real environments while avoiding collisions with obstacles and falling off cliffs. By normalizing the proximity and ground sensor readings and training in randomly generated environments, the controller learns policies that maximize spatial coverage, avoid repeated visits to the same locations, and favor smooth, forward movement. The results showed that more complex algorithms aren't always desirable for simple problems like the one presented, achieving more unreliable results with the implementation of memory in the PPO and an ANN in the line following problem.

Keywords: Braitenberg Machine, Evolutionary Algorithms, Neural Networks, Reinforcement Learning, Proximal Policy Optimization (PPO)

1 Introduction

This project was developed in two distinct parts, with the common goal of creating an autonomous controller for the Thymio robot, capable of exploring environments while avoiding collisions with obstacles and falls from cliffs.

In the first part, more traditional control methods were explored, based on evolutionary algorithms and artificial neural networks (ANNs). Initially, an approach based on Braitenberg machines was implemented, using simple connections between sensors and actuators to generate reactive behaviors. Subsequently, simple neural networks were developed, using only the ground sensors. The next step involved the development of more advanced neural networks that combined ground sensors with front-facing proximity sensors, allowing for obstacle avoidance. These approaches enabled the study of the robot's response to different sensory configurations and simple control strategies.

In the second part of the project, we explored the use of Deep Reinforcement Learning algorithms, specifically the Proximal Policy Optimization (PPO) algorithm, to train a more sophisticated controller. The goal was to enable the robot, starting from the central area of the environment and with any initial orientation, to efficiently explore the space while avoiding collisions and cliff falls. To achieve this, a neural network was used that, based on sensor observations — five front-facing proximity sensors and two downward-facing ground sensors — learned to determine the speed of each robot wheel in order to maximize safe and effective exploration performance.

This integrated approach allowed for a comparison between classical and modern robotic control techniques, evaluating their robustness, generalization ability, and effectiveness in autonomous navigation in complex environments.

2 Methodology

This section focuses on the approaches implemented to develop effective autonomous behavior in the Thymio robot. Both classical solutions based on evolutionary algorithms and artificial neural networks, as well as modern techniques involving Deep Reinforcement Learning, were explored.

2.1 Part A: Classical approaches based on evolutionary algorithms and artificial neural networks

In the first phase of the project, classical behavioral control approaches were implemented and tested using evolutionary algorithms and artificial neural networks (ANNs).

The experiments in Part A were conducted in a fixed, predefined simulation environment featuring a black line path over a white floor and optionally static obstacles. This environment was specifically designed to test basic reactive behaviors and assess line-following and obstacle avoidance capabilities under controlled conditions.

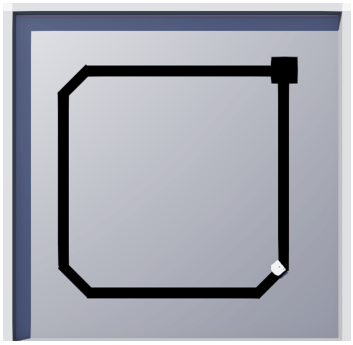

Figure 1: Simulation environment used for Part A (Evolutionary approaches).

2.1.1 Phase 1: Evolutionary Braitenberg Machine

The objective of this phase was to develop a simple reactive controller based on the concept of a Braitenberg Machine, with its parameters optimized using an Evolutionary Algorithm (EA).

The controller architecture consists of two ground sensors as input (left and right) and two outputs corresponding to the left and right motor speeds. The motor speeds are calculated through linear equations that combine the sensor readings with weighted parameters, as follows:

$$\text{Left\_Motor} = (P_{1L} \cdot S_L) + (P_{2L} \cdot S_R) + P_{3L} \quad (1)$$

$$\text{Right\_Motor} = (P_{1R} \cdot S_L) + (P_{2R} \cdot S_R) + P_{3R} \quad (2)$$

, where S_L and S_R represent the readings from the left and right ground sensors, respectively. In total, there are six parameters to be optimized: three for the left motor (P_{1L}, P_{2L}, P_{3L}) and three for the right motor (P_{1R}, P_{2R}, P_{3R}).

These parameters were evolved with the goal of maximizing the robot's performance in the exploration task, keeping it in motion and avoiding cliff falls, relying solely on the information provided by the ground sensors.

Initially, a population of controllers with randomly generated weights was created. For each generation, the evolutionary process involved the following steps:

- Each individual controller was evaluated by simulating its behavior and measuring its performance, expressed as fitness.
- The best-performing individuals were selected to act as parents for the next generation.
- New individuals were generated from these parents using crossover and mutation operations.
- The average fitness of the population for the generation was recorded.

The fitness function was designed to reward behaviors that keep the robot over the black line. At each simulation step, the readings from the two ground sensors were analyzed. The function increased the fitness score whenever the robot was correctly positioned on the line — with greater rewards if both sensors detected the black line simultaneously. The total fitness of an individual is the accumulated score over the evaluation time, normalized as:

$$fitness = \frac{Time\ on\ Line}{Evaluation\ Time} \quad (3)$$

This encourages solutions that can consistently follow the line throughout the simulation.

The parameters used in the genetic algorithm were:

- **Population size**: 10 individuals
- **Generations**: 300
- **Elitism**: The top 3 individuals (based on fitness) are preserved in each generation
- **Crossover**: One-point crossover between parent weight vectors
- **Mutation**: Each gene has a 20% chance of being mutated with Gaussian noise ($\sigma = 0.05$)
- **Genome representation**: 6 floating-point weights, corresponding to the motor control equations

The evolutionary algorithm optimized the parameters over generations, producing a Braitenberg-style controller capable of navigating by following the black line using only ground sensor inputs.

2.1.2 Phase 2: Simple ANN (Ground Sensors Only)

The objective of this phase was to evolve a minimalistic artificial neural network (ANN) for a complex task involving both initial exploration and subsequent line-following, constrained to using only the robot's two ground sensors.

The network architecture was strictly defined as follows:

- **Inputs:** Two normalized values from the ground sensors.
- **Hidden Layer:** Four neurons with hyperbolic tangent (*tanh*) activation.
- **Outputs:** Two neurons, also with *tanh* activation, controlling the left and right motor speeds.

The network's 22 parameters, encompassing all weights and biases, were subject to evolutionary optimization:

$$\underbrace{(2 \times 4)}_{\text{input-to-hidden weights}} + \underbrace{4}_{\text{hidden biases}} + \underbrace{(4 \times 2)}_{\text{hidden-to-output weights}} + \underbrace{2}_{\text{output biases}}$$

Raw sensor values were normalized to the range $[-1, 1]$. The network outputs were then scaled to motor speeds to ensure continuous forward motion using the formula:

$$\text{speed} = \text{BASE\_SPEED} + \text{output} \times (\text{MAX\_SPEED} - \text{BASE\_SPEED}) \quad (4)$$

where BASE_SPEED = 3.0 and MAX_SPEED = 6.28.

A primary challenge in this phase is the robot's "perceptual blindness" when off the line, as the sensors provide no directional information for search. To guide the evolution, a **shaped fitness function** was employed, consisting of two components:

- **Execution Fitness (F_{exec}):** This component rewards the **distance traveled** while on the line, weighted by a multiplier of 1.0 for being centered (both sensors on the line) and 0.5 for partial contact (one sensor on the line).
- **Exploration Fitness (F_{expl}):** A fixed bonus with a maximum value of 10.0 is awarded for **finding the line quickly**. The bonus is scaled by the proportion of evaluation time remaining after the first contact is made.

The total fitness for an individual is the sum of these two components:

$$\text{Fitness}_{\text{total}} = F_{\text{exec}} + F_{\text{expl}} \tag{5}$$

The network parameters were evolved using a Genetic Algorithm (GA) featuring an **adaptive mutation strategy**. This strategy was designed to balance broad exploration in early generations with fine-tuning in later stages. The GA was configured as follows:

- **Population Size:** 30 individuals
- **Generations:** 300
- **Selection:** The top 4 individuals are selected as parents for reproduction.
- **Elitism:** A single-best elitism strategy was used, where the top-performing individual from each generation is guaranteed to pass, unaltered, to the next.
- **Crossover:** Single-point crossover between two randomly selected parents from the elite group.
- **Adaptive Mutation:** Both the mutation rate and size decrease linearly over the first 75% of generations. The mutation rate anneals from 0.25 to 0.05, and the mutation size (standard deviation of Gaussian noise) anneals from 0.5 to 0.1.

To ensure the development of robust and generalizable behaviors, the robot's starting position and orientation were fully randomized within the arena before each 200-second evaluation period. This methodology combines a simple reactive controller with a sophisticated evolutionary optimization process to address the challenges of learning with limited sensory information.

2.1.3 Phase 3: Advanced Artificial Neural Network (Ground Sensors + Obstacles)

The network architecture was expanded to include five inputs: two ground sensors and three front proximity sensors. The hidden layer consisted of 4 neurons with hyperbolic tangent (*tanh*) activation, and the output layer had 2 neurons corresponding to the left and right motor speeds. The increased number of inputs raised the total parameters subject to optimization to 34, calculated as:

$$\underbrace{5 \times 4}_{\text{input-to-hidden weights}} + \underbrace{4 \times 2}_{\text{hidden-to-output weights}} + \underbrace{4 + 2}_{\text{biases for hidden and output layers}}$$

To harmonize sensor data, raw proximity sensor readings were linearly normalized to the $[0, 1]$ interval, consistent with the normalization applied to ground sensors. This enabled the network to integrate terrain and obstacle information effectively.

Given the increased task complexity, the evolutionary algorithm parameters were adjusted: the population size was increased, and the number of generations extended to facilitate convergence. The fitness function was enhanced to evaluate both line-following performance and obstacle avoidance, penalizing collisions to promote safe navigation.

The evaluation environment incorporated fixed obstacles along the track, simulating real-world challenges. Performance metrics accounted for the robot's ability to maintain line tracking without collisions and encouraged smooth navigation trajectories, guiding the evolution of more sophisticated control behaviors.

To guide the evolutionary process, a multi-objective fitness function was implemented, balancing three key performance indicators: line-following accuracy, obstacle avoidance, and area coverage. The function is defined as:

$$fitness = w_{line}\dot{S}_{line} + w_{obstacle}\dot{S}_{obstacle} + w_{area} + S_{area} - P \tag{6}$$

, where:

- S_{line} is the proportion of time the robot remained on the line,
- $S_{obstacle}$ is the proportion of time without collisions
- S_{area} is the ratio of distance covered to the maximum possible distance,
- w_{line}, $w_{obstacle}$, w_{area} are the respective weights for each objective,
- P is the total penalty applied for poor performance in any of the objectives.

Penalties were introduced to discourage undesirable behaviors:

- A collision penalty was applied if the number of collisions exceeded a predefined threshold.
- An off-line penalty was triggered if the robot spent less than 5% of the time on the line.
- An area penalty was applied when the robot covered less than 20% of the expected distance.

All scores were clamped to the [0,1] interval to ensure stability and comparability across different runs. The final fitness value was also clamped to a minimum of zero to prevent negative values from disrupting the evolutionary process.

2.2 Part B: PPO Implementation

In the second part of the project, a Deep Reinforcement Learning (DRL) approach was adopted using the Proximal Policy Optimization (PPO) algorithm. The objective was to train a more robust and generalizable policy capable of navigating a dynamic environment, avoiding both obstacles and cliff edges.

The environment, shown in Figure 2, was designed with a static H-shaped track layout but featured dynamically positioned obstacles. This setup, known as *domain randomization*, forces the agent to learn general strategies for obstacle avoidance rather than memorizing a single static path.

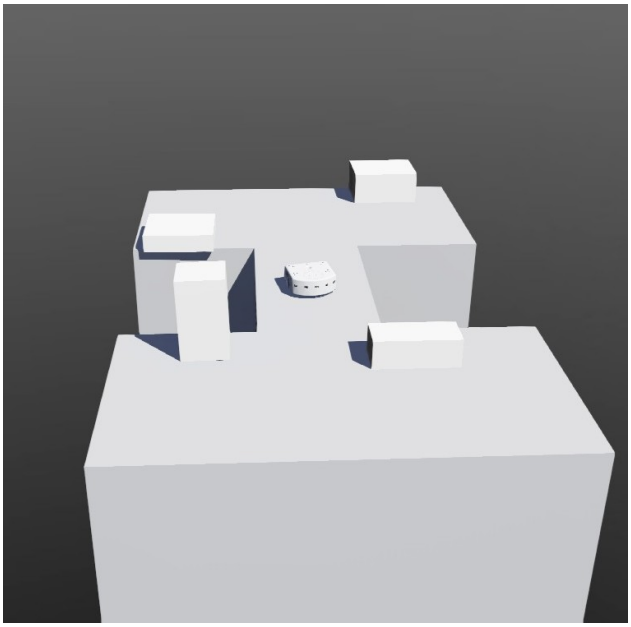

Figure 2: Simulation environment used for training the PPO agent, featuring an H-shaped track and randomized obstacle placement.

2.2.1 Reinforcement Learning Framework

The problem was framed within the standard reinforcement learning paradigm, where an agent interacts with an environment to maximize a cumulative reward signal.

- **State and Action Spaces** The agent's perception and interaction with the world were defined as follows:
  - **Observation Space:** A 7-dimensional continuous vector containing normalized sensor readings. This included five frontal proximity sensors to detect obstacles and two ground sensors to detect the track edges. Each sensor value was normalized to a range of $[0, 1]$.
  - **Action Space:** A 2-dimensional continuous space, where each value in the range $[-1, 1]$ corresponds to the normalized velocity for the left and right motors, respectively.
- **Policy Network Architecture** The agent's "brain" is a policy network implemented as a Multi-Layer Perceptron (MLP). The PPO algorithm, sourced from the Stable-Baselines3 library, was used to train this network. The architecture was configured as follows:
  - **Input Layer:** 7 neurons, corresponding to the observation space.
  - **Hidden Layers:** Two hidden layers with 64 and 32 neurons, respectively, using the ReLU and Tanh activation functions.
  - **Output Layer:** 2 neurons, producing the continuous values for the action space.
- **Reward Engineering** The success of the DRL agent is critically dependent on the design of the reward function. A carefully shaped reward signal was engineered to encourage the desired emergent behaviors:
  - **Catastrophic Failure Penalty:** A large negative reward of -100 is given if the agent falls off the track. This is detected either by a low ground sensor reading or a significant drop in the robot's z-axis coordinate. This event immediately terminates the episode.
  - **Movement and Exploration Incentive:** A positive reward proportional to the distance traveled in each step ($+20\times$distance) encourages the agent to actively move and explore the environment, penalizing immobility.
  - **Proximity-Based Collision Avoidance:** A progressive negative penalty is applied as the robot gets closer to obstacles. The penalty is proportional to the sum of proximity sensor readings that exceed a threshold of 0.5, discouraging the agent from lingering near walls.
  - **Survival Bonus:** A small, constant positive reward of $+0.5$ is given at each step the agent survives without failure, incentivizing longevity.
  - **Episode Completion Bonus:** A large positive reward of $+200$ is awarded if the agent successfully survives for the entire maximum duration of an episode, strongly reinforcing robust, long-term survival strategies.
- **Training Process and Generalization** The training process was configured to promote robust and generalizable policies.
  - **Algorithm:** Proximal Policy Optimization (PPO).
  - **Hyperparameters:** A learning rate of 3×10^{-4} and an entropy coefficient (`ent_coef`) of 0.01 were used to balance policy optimization with exploration.
  - **Domain Randomization:** At the start of each episode, the positions, orientations, and sizes of the five obstacles were randomized. This prevents the agent from overfitting to a single environment configuration and forces it to learn a more general obstacle avoidance policy.

- **Normalization:** The `VecNormalize` wrapper from Stable-Baselines3 was used to normalize both observations and rewards during training, a standard technique for stabilizing the learning process in DRL.
- **Training Duration:** The agent was trained for a total of 100,000 timesteps, with model checkpoints saved periodically.

2.2.2 Phase 2: Recurrent Policy Optimization

Building upon the foundation of the first DRL phase, the second phase addressed a key limitation of the standard feedforward policy: its lack of temporal memory. While the 'MlpPolicy' reacts only to the immediate sensory input, many real-world navigation challenges benefit from understanding the recent history of observations and actions. To this end, a recurrent policy was implemented.

The core change in this phase was the adoption of the **Recurrent Proximal Policy Optimization (RecurrentPPO)** algorithm, a variant of PPO designed to train policies with memory. This was paired with a corresponding recurrent network architecture.

- **Recurrent Policy Network (*MlpLstmPolicy*)** The key innovation in this phase is the introduction of a **Long Short-Term Memory (LSTM)** layer into the policy network. The *MlpLstmPolicy* architecture maintains a hidden state that is updated at each timestep, allowing it to integrate information over time. This enables the agent to learn from sequences of observations, rather than just instantaneous snapshots.
- **Environment and Reward Structure** The underlying simulation environment, including the state/action spaces and the carefully engineered reward function, remained identical to the one used in the first PPO phase. This controlled setup ensures that any observed differences in performance can be directly attributed to the change in the policy's architecture—from a reactive *MlpPolicy* to a memory-aware *MlpLstmPolicy*.
- **Training Configuration** The training process was adapted to accommodate the recurrent nature of the model, while keeping other key parameters consistent for a fair comparison:
  - **Algorithm:** Recurrent Proximal Policy Optimization (RecurrentPPO), from the `sb3-contrib` library.
  - **Policy Architecture:** 'MlpLstmPolicy', which incorporates an LSTM layer alongside the MLP feature extractor.
  - **Hyperparameters:** The learning rate (3×10^{-4}) and entropy coefficient (0.01) were kept consistent with the previous phase.
  - **Training Duration:** The total number of timesteps was increased to 300,000 to provide sufficient time for the more complex recurrent policy to converge.

This phase aims to demonstrate whether the introduction of recurrent memory can lead to a more stable, intelligent, and robust navigation policy compared to the purely reactive agent from the preceding experiment.

3 Results

Here we analyze and discuss several different results for different parameters of the different algorithms, any relevant graphs not included here have been added to the Appendix (see Appendix A)

3.1 Evolutionary Braitenberg Machine

The application of the Evolutionary Braitenberg Machine was very successful in creating an agent capable of following the line.

There is a clear overall increase in fitness across generations, indicating that the evolutionary algorithm progressively improves the controllers' ability to complete the task (staying on the line) and gaining better fitness.

As per the requirements, the robots were trained with two sets of parameters for comparison:

- Parameters1, as in Fig. 3

```
# ...
MUTATION_RATE = 0.2
MUTATION_SIZE = 0.05
# ...
```

- Parameters2, as in Fig. 4

```
# ...
MUTATION_RATE = 0.3
MUTATION_SIZE = 0.1
# ...
```

Where MUTATION_RATE controls if a gene (a single weight) mutates, while MUTATION_SIZE controls how much a gene mutates.

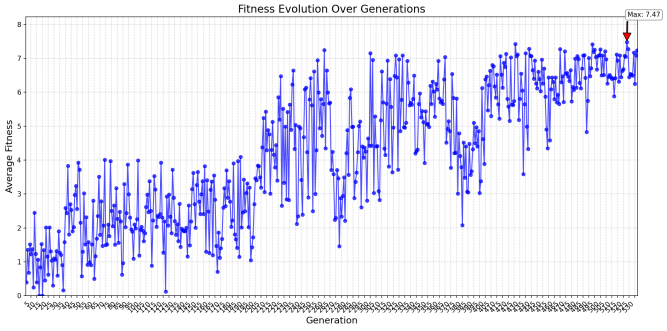

Figure 3: Evolutionary Algorithm: Average Fitness Across Generations – Parameters 1

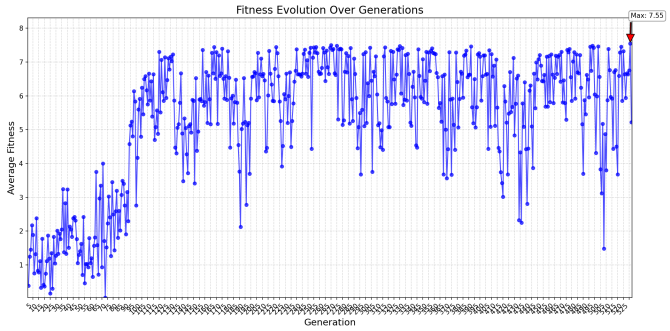

Figure 4: Evolutionary Algorithm: Average Fitness Across Generations – Parameters 2

As we can see, having a higher mutation rate and mutation size made it possible for our agent to learn much faster (it stagnated almost 300 generations before Fig. 3!), this shows just how much tuning small parameters can change the behavior and performance of the robot during and after training.

Despite this, the evolutionary method still exhibits a lot of variation in the average fitness accross generations, this can be attributed to factors like:

- Random initial starting position and direction of the robot, witch influences it's fitness.
- Mutation and crossover operators introduce diversity, which can lead to both improvements and regressions in fitness.
- Small variations lead to big differences in the time spent in the line and thus the fitness of the robot.

3.2 Simple ANN Results

The evolution of the simple ANN controller was monitored over 300 generations. The results, depicted in Figure 5 and Figure 6, reveal a distinct two-phase evolutionary process: a period of rapid initial learning followed by a phase of high volatility and performance stagnation.

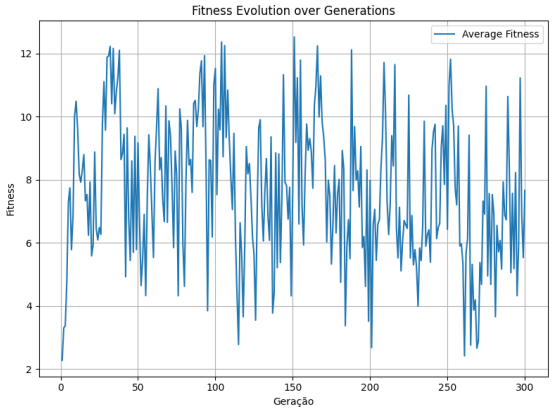

Figure 5: Average fitness evolution over 300 generations. The graph shows high volatility and a lack of consistent long-term improvement after an initial learning phase.

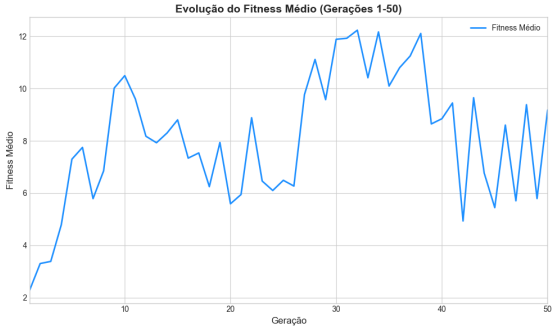

Figure 6: A focused view of the first 50 generations, highlighting the rapid and consistent increase in average fitness as the population first learns to solve the exploration task.

- **Rapid Initial Learning (Generations 1-50)** As shown in Figure 6, the initial phase of evolution is characterized by a steep and consistent rise in average fitness, increasing from approximately 2.0 to peaks exceeding 12.0. This rapid improvement demonstrates the effectiveness of the shaped fitness function. The large exploration bonus (F_{expl}), combined with the high initial parameters of the adaptive mutation strategy, strongly incentivized the discovery of any behavior that resulted in finding the line. During this stage, the primary evolutionary pressure was to solve the exploration problem, which the genetic algorithm achieved successfully.

- **Performance Volatility and Stagnation (Generations 50-300)** Following the initial learning phase, the global view in Figure 5 shows that the average fitness fails to converge to a stable, high value. Instead, it exhibits extreme volatility, oscillating between high peaks and deep troughs. This behavior reveals a fundamental conflict between the two objectives of the task: exploration and execution.

  The high-scoring individuals evolved into "specialists" proficient at line-following (F_{exec}). However, these specialized behaviors were often ineffective for exploration. Due to the randomized starting positions, these specialists would frequently fail to find the line at all, resulting in a fitness score of zero and causing the population's average fitness to plummet in subsequent generations.

  Ultimately, this volatility is a direct consequence of the ANN's architectural simplicity. With only two inputs and no internal memory, the network is incapable of context-aware behavior; it cannot differentiate between being "lost far from the line" and "just falling off the line" to switch between searching and following strategies. The resulting reactive behaviors are inherently brittle, leading to the observed performance instability. The results do not signify a failure of the evolutionary process, but rather successfully characterize the performance limitations of a simple reactive controller on a complex task.

3.3 Advanced ANN (Ground Sensors + Obstacles)

For the advanced ANN, 3 proximity sensors were used we choose sensors 0, 2, and 4 and we verified for collisions with 3 states if below 3000 for no collisions, between 4000 > value > 3000 for near a collision and if > 4000 we assumed there was a collision, to monitor for collisions we used all proximity sensors with value > 4300, for the line we used a similar method to the simple ANN where is multiplies a reward with the distance walked before last step, then we just add the sum of the black line fitness with the proximity sensor, for the rewards we found that a very high reward for the line helps a lot and then when the training starts to be fitter it improves smaller steps, for the line the reward was 50 if both sensors are in the black line, and 40 if only one of the sensors, for the proximity sensors we have 3 states for the when there was a collision we subtract 10, with no collision we added 2 and near a collision we added 1. For the advanced ANN we performed 2 tests with the same configuration except for the number of neurons, one with 4 and another with 10 neurons, for these tests, we also set the mutation size 0.05 and the mutation rate to 0.2, and the robots ran for 300 generations, with 300 for evaluation time. For these tests we find it harder to learn due to the unpredictability of the random objects, a good run could easily be a bad run in the next evaluations, some of the initial tests were also learning from upside-down behavior, where the robot after some generations was throwing itself against the objects to go upside down and get a perfect reading for the color sensors, we think an accelerometer could help a lot in this situation, but this also happen because of some rotation configurations not being perfectly adjusted.

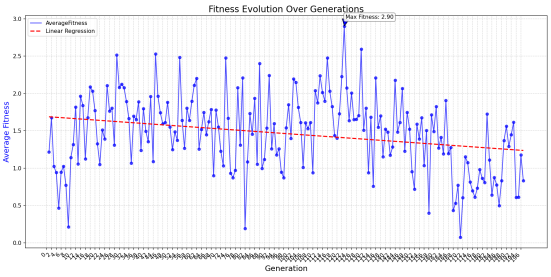

Figure 7: Average Fitness evolution over 300 generations for 4-neuron network

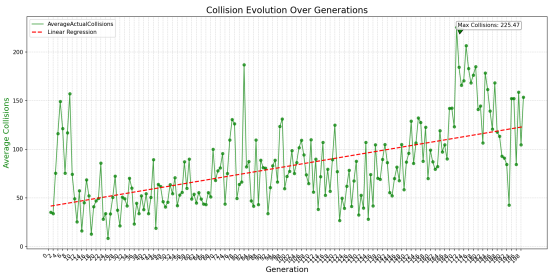

Figure 8: Collision count over 300 generations for 4-neuron network

For the test with the 4 neurons it did not perform very well as we can see the figure 7 where the fitness dropped as the collisions got higher 8, we believe this happened because because it did not learn soon enough to avoid objects in the line, or avoid objects while traveling to the line, for the 10 neurons the test performed better as we can see in 7 where it had a inverse result, the collisions got lower as demonstrated in the figure 10 and the fitness improved over time 9, confirming our expectation that the 10 neurons network would perform better in this conditions.

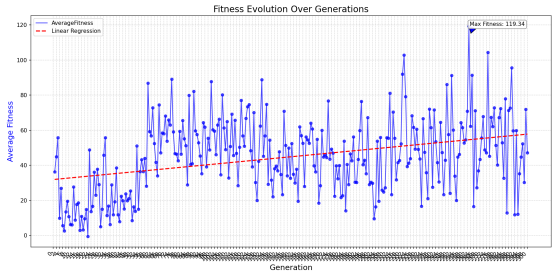

Figure 9: Average Fitness evolution over 300 generations for 10-neuron network

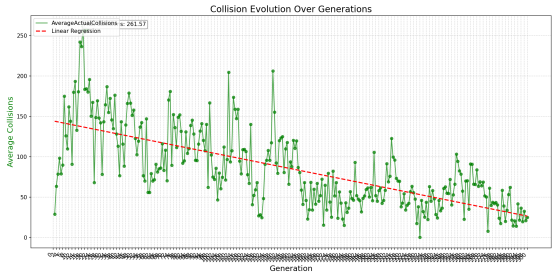

Figure 10: Collision count over 300 generations for 10-neuron network

3.4 Part B: PPO Implementation

The performance of the Deep Reinforcement Learning agents was evaluated comparing the standard Proximal Policy Optimization (PPO) with its recurrent variant (RecurrentPPO). The analysis focused on the impact of the reward structure, activation functions (ReLU vs. Tanh), and environment randomization.

3.4.1 PPO Performance with ReLU Activation

The training results for PPO agents using the ReLU activation function are presented in Figure 11. The configuration with penalties and randomized obstacles (`PPO_ReLU_randObs_penalty`) achieved the highest average reward, outperforming all other variants. In contrast, configurations without penalties (`_noPenalty`) learned suboptimal policies, often maximizing reward by remaining stationary, which highlights the necessity of a well-shaped reward function for meaningful learning.

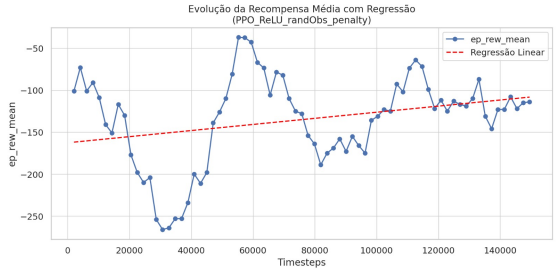

Figure 11: Fitness evolution over generations: evolutionary Braitenberg machine

3.4.2 PPO Performance with Tanh Activation and Comparative Insights

Figure 12 shows the performance of PPO agents using the Tanh activation function. While these agents achieved reasonable performance (`fixedObs`), their effectiveness degraded significantly when faced with randomized obstacles, meaning that the tanh function, in this context, was less capable of producing a generalizable policy.

This shows that the best-performing model for was the one combining PPO with ReLU, with a penalty system, and domain randomization.

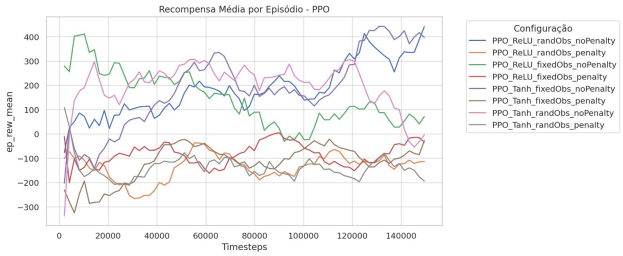

Figure 12: Fitness evolution over generations: evolutionary Braitenberg machine

3.4.3 Comparative Analysis with RecurrentPPO

RecurrentPPO (RPPO) was also used, the obtained results can be seen in Figure 13, but the correspond to the contrary of the initial hypothesis.

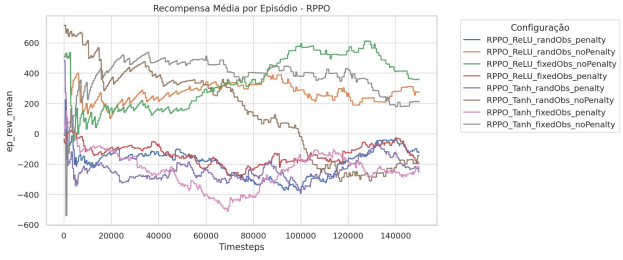

Figure 13: Fitness evolution over generations: evolutionary Braitenberg machine

The RPPO agents exhibited much greater training instability and failed to consistently outperform their simpler, non-recurrent counterparts.

This outcome suggests that for this navigation task, the additional complexity of the recurrent network was not beneficial and may have even hindered sample efficiency. The purely reactive `MlpPolicy` of the standard PPO was sufficient to learn an effective policy from the sensory information.

4 Conclusion

This project demonstrated that while both classical and modern approaches can yield effective controllers for Thymio robots, their performance depends on task complexity and algorithm suitability. Evolutionary algorithms and simple ANNs performed well in structured environments, but struggled with generalization. Deep Reinforcement Learning, particularly PPO with ReLU and well-shaped rewards, achieved robust performance in randomized settings. However, adding memory through Recurrent PPO introduced instability, showing that more complex models are not always advantageous for relatively simple navigation tasks.

Appendix A Additional Graphs

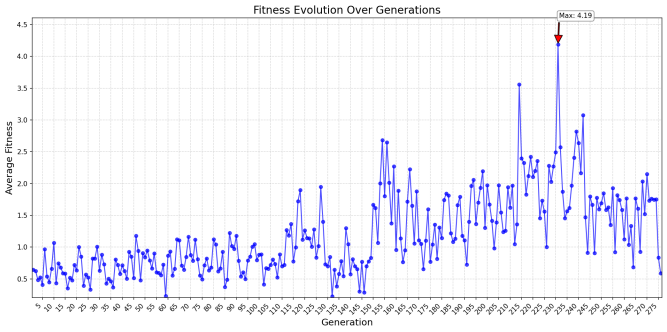

Figure A.14: Simple ANN, where fitness of each individual was calculated as an average of 3 runs of that individual.

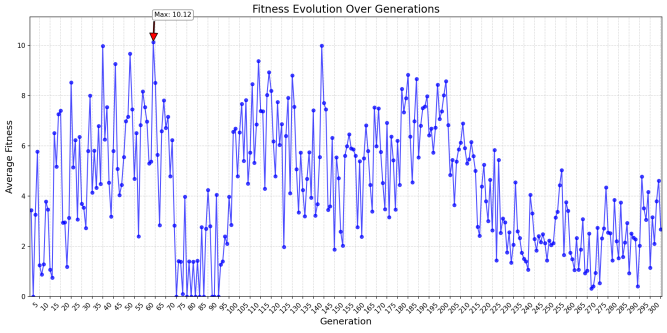

Figure A.15: Simple ANN with adaptive fitness to encourage exploration in the beginning and exploitation in the end.

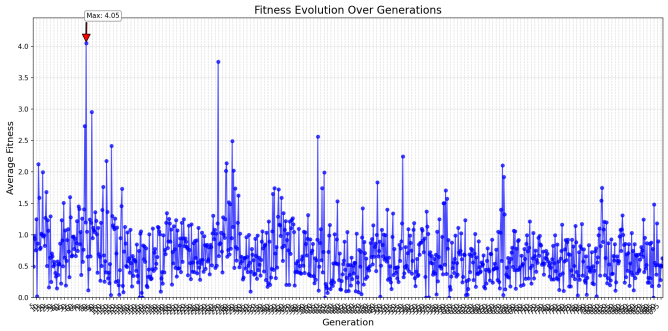

Figure A.16: Earlier simple ANN iteration

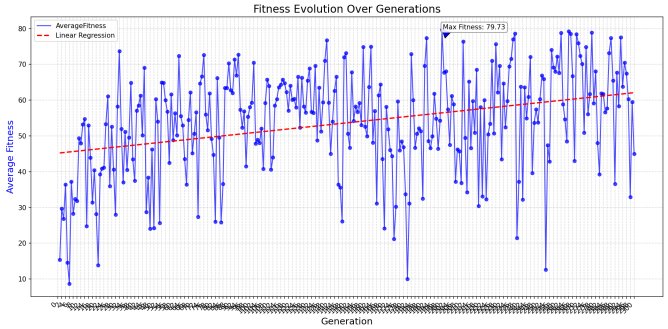

Figure A.17: Earlier advanced ANN iteration, before calculation of distance between steps