Autonomous Robotic Systems: Assignment

Due on 9 March 2021

Rico Möckel

Frederic Abraham - i6262598
Theodoros Giannilias - i6255101
Guillaume Whatever - i6256477

Description of our Genetic Algorithm and hyper-parameters used

Artificial Neural Networks

The architecture of the ANN that we are using is visualized in figure 1.

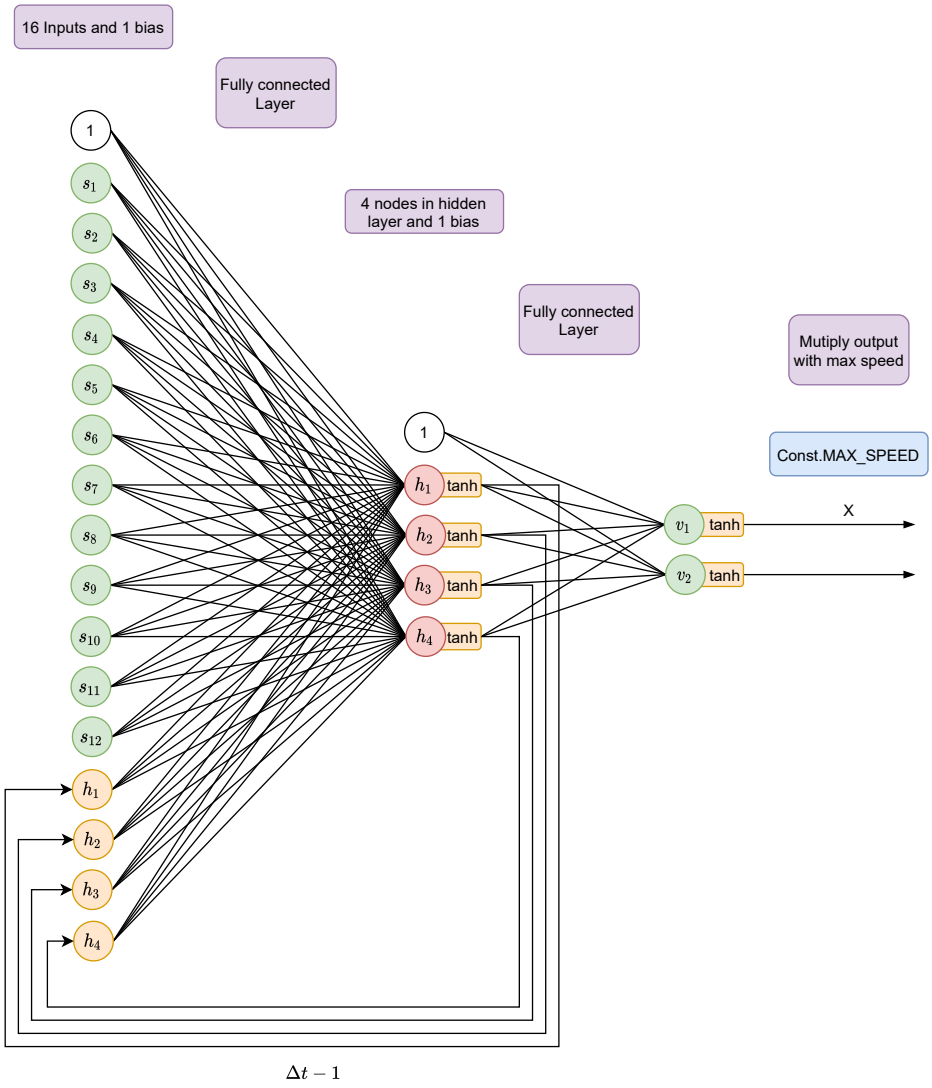
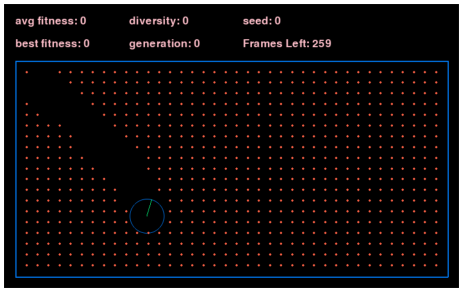


Figure 1: Architecture of ANN

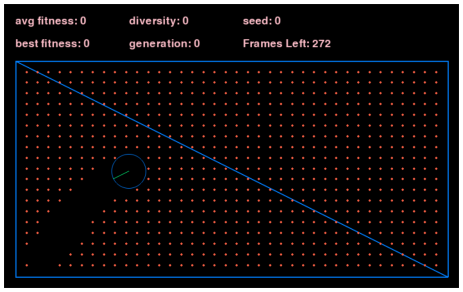
We use the *HyperbolicTangent* function for both the hidden and the output layer. Because of that, the value obtained in the output layer belong to the interval $[-1, 1]$ that we can consider as the percentage of velocity for each wheel and are thus multiplied by a fixed value named **MAX_SPEED**.

Morphology

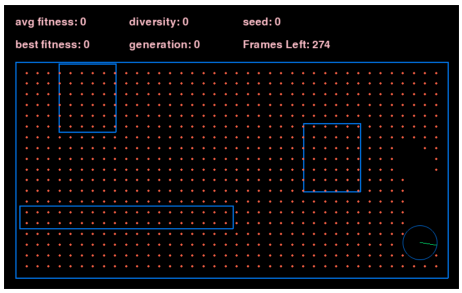
We are using our own simulator created in assignment 2 with a few modifications in order to calculate the fitness appropriately. We added different kind of rooms to be able to train a more stable robot behaviour. The rooms can be seen on the next page.
The first iterations of training was done with every generation on a new room. That led to promesing genomes that done well in a more spacious room dieing in the smaller rooms and the avg and best fitness



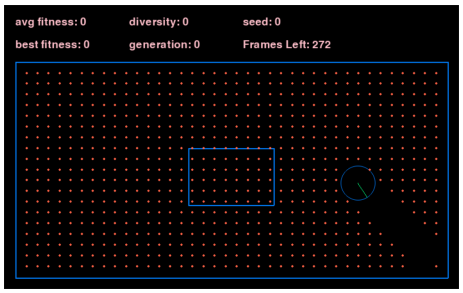
(a) Room 1: Empty Room



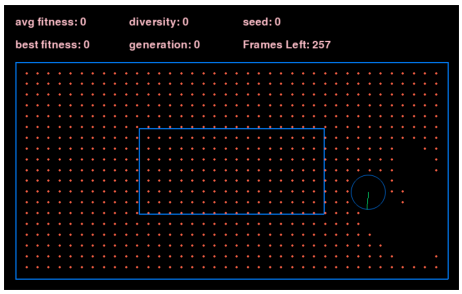
(b) Room 2: Triangle



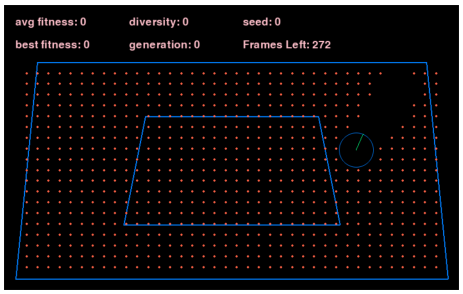
(c) Room 3: Obstacles



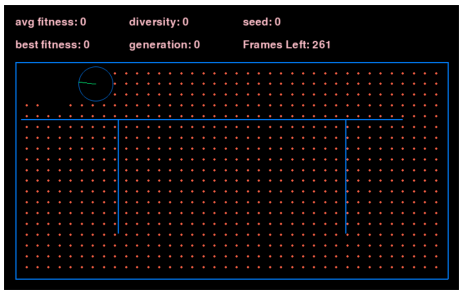
(d) Room 4: Small Box



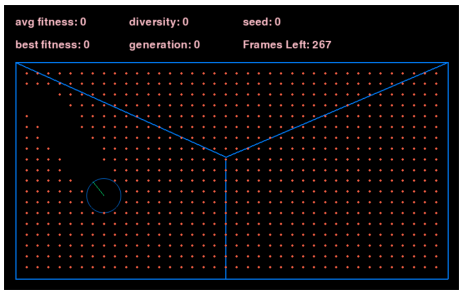
(e) Room 5: Bigger Box



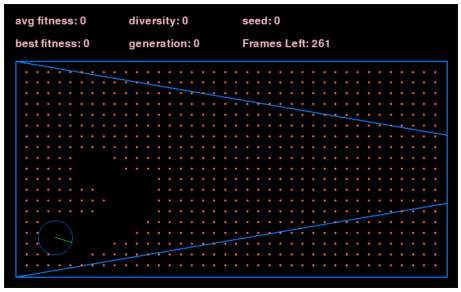
(f) Room 6: Trapazoid



(g) Room 7: Labyrinth



(h) Room 8: Propeller



(i) Room 9: Long Slope

didn't show a reliable trend.

Genetic Representation

The genetic representation used consisted in a sequence of floats where each element represents a weight for a connection input-hidden or hidden-output. The list is therefore divided in 2 sections, the first one contains the weight for the input-hidden connections, the second one the hidden-output ones as shown in Figure 3:

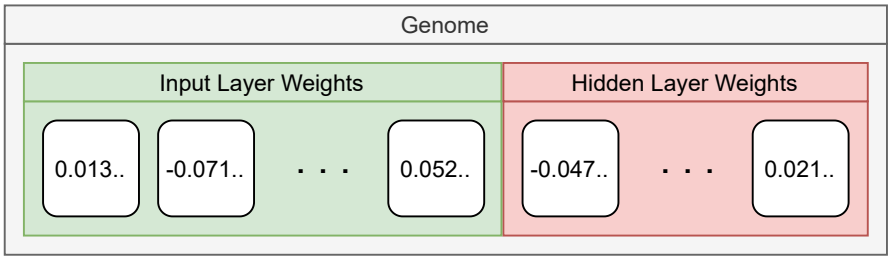


Figure 3: First 21 genes of a genome

Population Size

The initial population size that we are using in our Genetic Algorithm is 50 individuals.

Initialization

The initialization process is straight forward, we create a list of random floats generated using a uniform distribution with boundaries. Specifically the interval used for the genome values is $[-1, 1]$, this has been done to follow the usual protocol of initialization of the weights for Neural Networks.

Exact Fitness Function

Different approaches were tried:

1. The first one reflects the first stop criterion:

$$fitness = (\frac{time\_survived}{maximum\_time} \times \frac{particles\_collected}{total\_particles}) \times 100 \tag{1}$$

2. The second one instead is used with the respective stop criterion:

$$fitness = \frac{(\frac{1}{1+collisions} + 2 \times \frac{particles\_collected}{total\_particles}) * 100}{3} \tag{2}$$

Stop Criterion

For this task two stop criteria have been explored:

- **N. Collisions**: The first criterion stops the update of the individuals once a wall has been hit. There are two main advantages with this approach. The first one consist of a reduction of the computational time, in fact we're not really interested in individuals that hit the walls. The second one is the fact that the robots tend to learn pretty quickly not to hit the wall, which is a big part of the task. There is, though a downside to this, the exploitation is used way more than the exploration, leading only to the discovery of local minimum possibly.

- **Life Steps**: This second criterion leaves more space for the exploration and defines a limited time for the robots to move around the room and try to collect as much dust particles as possible. It is important to mention that this is also is part of the first criterion, in fact, in case a robot keeps on moving without hitting the walls, the life counter is decremented each update to ensure a limited time for each generation.

Selection and Reproduction

In this section, we provide the Selection and Reproduction strategies we implemented in our Genetic Algorithm:

1. Selection Strategies:
    - Tournament
    - Rank-Based
    - Roulette-Wheel
2. Reproduction Strategies:
    - Crossover
    - Mutation
    - Elitism

Crossover and Mutation Operators

Below a list of the different Crossover and Mutation Operators that we decided to implement and test:

1. Crossover Operators:
    - Single-Point
    - Double-Point
    - Multi-Point
    - Uniform
    - Linear
    - Arithmetic
2. Mutation Operators:
    - Standard
    - Integer-Transform
    - Bit-Flip
    - Swap
    - Boundary
    - Gaussian