Evolutionary Robotics

Assignment 2: Hill Climbers

Prof. Dr. Javad Ghofrani

Winter Semester 2025

November 3, 2025

Amritanshu Amrit
Bhavesh Gandhi

Contents

1 Task 1: Hill Climber – Text Evolution

1.1 a) Implementation Output

The program was implemented in Python. A typical run produces output where the fitness (percentage of correct characters) gradually increases until it reaches 100%. A sample of the output from a single run is attached in the zip.

1.2 b) A Good-Natured Optimization Problem

This problem is considered "good-natured" for a hill-climbing algorithm because its fitness landscape is exceptionally smooth and devoid of local optima.

- **No Local Optima:** In this problem, for any string that is not the perfect target, there is always a path to a better solution. If a string has a fitness less than the maximum (33), at least one character is incorrect. A single mutation at that incorrect position can change it to the correct character, which will increase the fitness by 1. The hill climber will never get permanently "stuck" on a sub-optimal solution.
- **Smooth Fitness Gradient:** Every time a correct character is found, the fitness increases. This creates a smooth gradient that the hill-climber can easily follow uphill toward the optimal solution.

1.3 c) Mathematical and Empirical Estimation

1.3.1 Mathematical Estimation

Let $L = 33$ is the string length and $C = 27$ is the size of the character set. If we already have k correct characters, the probability p_k of improving the string in one generation is the probability of picking one of the $L - k$ incorrect positions and mutating it to the correct character:

$$p_k = \frac{L-k}{L} \times \frac{1}{C}$$

The average number of generations to get this one improvement is $1/p_k$. To find the total average generations, we sum the time it takes to find each correct character, from the 1st to the 33rd:

$$E_{\text{total}} = \sum_{k=0}^{L-1} \frac{1}{p_k} = \sum_{k=0}^{L-1} \frac{L \cdot C}{L-k} = (L \cdot C) \sum_{j=1}^{L} \frac{1}{j}$$

Since this is a Harmonic Progression, the final sum is the L-th Harmonic Number, H_L.

$$H_L = \frac{1}{d} ln \frac{2a + (2n-1) \cdot d}{2a - d}$$

where **a=1** is the first term, **d=1** is the common difference of the corresponding AP, and **n=33** is the number of terms

$$E_{\text{total}} = L \cdot C \cdot H_L \approx 33 \cdot 27 \cdot (\ln(67)) \approx 891 \cdot 4.204 \approx \mathbf{3746}$$

Our mathematical estimation is approximately **3746 generations** on average.

1.3.2 Empirical Estimation

To test this prediction, the simulation was run 1000 times, and the number of generations required for each run was recorded.

```
--- Results ---
Number of runs: 1000
Average generations needed: 3473.84
```

2 Task 2: Hill Climber for Robot Behavior

2.1 a) Implementation and Results

A hill-climbing algorithm was implemented in Python to evolve a controller for a simulated robot. The robot's goal is to explore as much of the arena as possible. The fitness is defined as the number of different grid cells visited during a fixed-time simulation. The robot's controller (genome) consists of 6 floating-point values representing the weights and biases of a simple linear controller mapping sensor inputs to wheel speeds.

After 5000 generations, the best-evolved controller produced the trajectory shown in Figure 1.

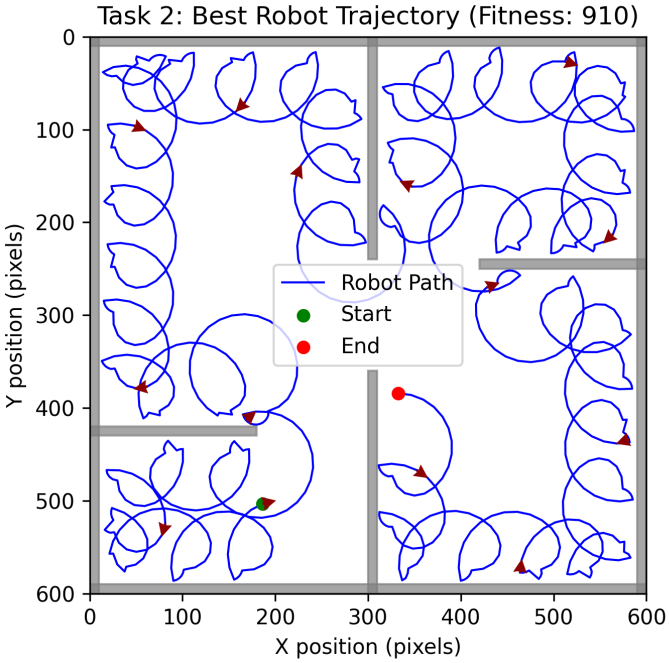

Figure 1: Trajectory of the best-evolved robot controller after 5000 generations.

2.2 b) Interpretation and Potential Improvements

2.2.1 Satisfaction with Results

The evolved behavior is surprisingly effective in maximizing exploration (though in a simple way) for such a simple controller and optimization algorithm.

- **Systematic Exploration:** The robot successfully navigates out of its starting corner and proceeds to explore all four main quadrants of the arena. The trajectory shows a clear, repeatable strategy. Within each quadrant, the robot follows a looping, spiral pattern, which allows it to cover the available space before moving to a new area.
- **Genome Initialisation:** The initial range of values for the genome and the values for the mutation change were arrived at after some testing. Too large a value could lead to higher jumps and miss the optima, whereas smaller values could take a lot of time to arrive at the optima.

However, the behavior is not perfect. While the looping is effective for coverage, it may not be the most efficient strategy for discovering new cells, as it spends time re-tracing nearby paths and incurs collisions with the wall.

2.2.2 Possibilities for Improvement

1. **Refined Fitness with Bonuses/Penalties:** The fitness function could be enhanced by introducing multipliers. A collision penalty (e.g., multiplying fitness by 0.5 on collision) would lead to smoother navigation. A displacement bonus, rewarding the straight-line distance between the start and end points, could encourage the robot to travel further across the arena instead of looping in one region.

2. **Speed Clamping:** The simulation should clamp the wheel speeds to a realistic maximum. Without clamping, the robot can achieve large speeds. This is physically unrealistic. Enforcing speed limits would promote the evolution of more robust and transferable controllers.

3. **Advanced Evolutionary Algorithm:** A simple hill climber is prone to getting stuck on local optima. The looping behavior might be a local optimum where small mutations do not lead to a better exploration strategy. Using a full Genetic Algorithm (GA) with a population of individuals and crossover would allow for a more diverse search of the solution space and could potentially escape these loops to find more globally optimal behaviors.

4. **More Complex Controller:** A more complex controller would allow for more complex relationships between sensor inputs and motor outputs. This could enable more sophisticated behaviors like recognizing specific environmental features and reacting accordingly.

Some of these improvements were implemented and tested out in code. But the final submission has these commented to portray the pure simple Evolutionary algorithm. Nevertheless, an example trajectory with fitness multipliers, both penal and reward, and a displacement bonus is shown in Figure 2.

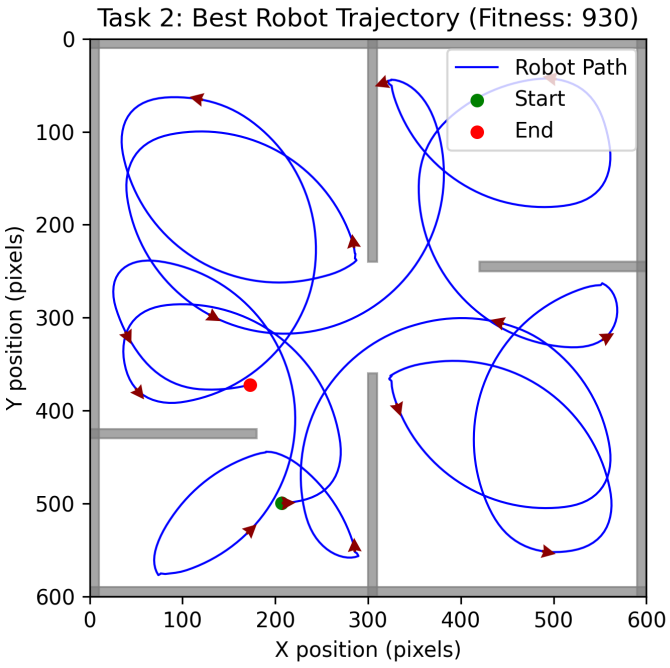

Figure 2: Trajectory, after improvement, of the best-evolved robot controller after 5000 generations.