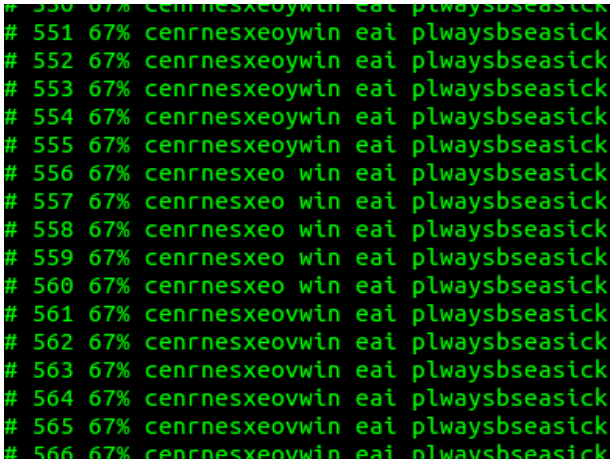
Task sheet 2 – Hill climbers

Deadline : 05.11.2025 23:59

Objectives:

- Implement a simple hill climber
- Understand the limitations of the hill climber strategy depending on the problem domain



1 Hill climber – evolutionary algorithms simplified

If artificial evolution is only a caricature of natural evolution then the following simple algorithm is only a caricature of evolutionary computation. Still, it is an interesting start because it is a straightfor- ward approach. We want to 'evolve' a certain string of text: "charles darwin was always seasick". We work not with a population but with just one individual and start with a random string composed of lowercase letters and spaces of correct length (33 characters). In each 'generation' we pick uniformly randomly one of the characters and replace it by a random character (lowercase letters and space). This implements our mutation. We accept this new string only if its fitness is not lower than before otherwise we backtrack to the former string. Hence, we implement a simple hill climber strategy. The fitness is just the number of correct characters (correct position and correct letter/space).

a) Implement this program. Print the current string in each generation.

b) One can say this is a good-natured optimization problem. Explain why this is plausible especially considering the implied fitness landscape.

c) Give a mathematical estimation of how many generations are needed in average to find the correct string. Test your predictions by determining an average empirically (produce many sample runs and average over the number of generations that were needed to find the correct string).

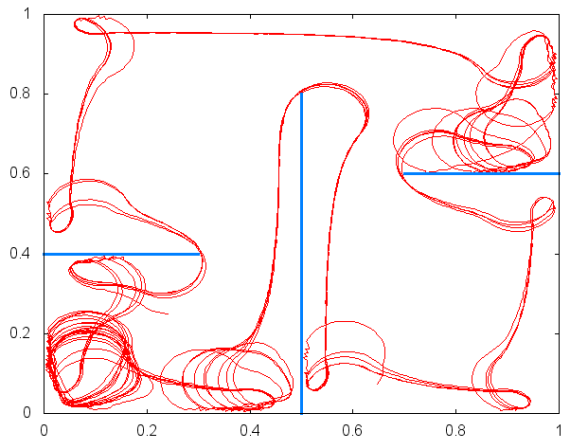
2 Hill climber for robot behavior

Now we utilize for the first time our simple robot simulator. The idea is to use a simple hill climber strategy to evolve a simple robot behavior. First we need an approach to encode different

behaviors (genetic representation). The robot has three proximity sensors and a differential drive as imple- mented in the last tutorial. Our approach is to implement reactive behaviors (no internal state/memo- ry), that is, the sensor input is just mapped to actuator outputs. Here we choose to have a linear map from sensor input to actuator output. The genetic representation is therefore pairs of two constants each: slope and intercept. We have three of these pairs, one for each sensor. The rotational speeds of the left (v_l) and the right wheel (v_r) are calculated based on the sensor input s (left s_l, middle s_m, right s_r) by

$$v_l = m_0 s_l + c_0, \tag{1}$$
$$v_r = m_1 s_r + c_1 + m_2 s_m + c_2, \tag{2}$$

whereas adding the sensor input of the middle sensor only to the right wheel speed is an arbitrary choice. We evolve a sequence of these three pairs $((m_0, c_0), (m_1, c_1), (m_2, c_2))$ which can be imple- mented, for example, as an array of floating point values. As above we have a population size of one and we start with a randomly generated genome (choose appropriate intervals for the parameters). You have to implement a loop that evaluates the current genome, that checks whether fitness has improved (keep genome) or worsened (backtrack to former genome), and that mutates the genome to produce a new candidate. The evaluation of a genome is done by placing the robot at a fixed position with fixed heading (for simplicity we have to enforce a deterministic evaluation here) in an environment as defined in the last tutorial (bounded area with walls) and then letting the robot run for a defined period of time. The fitness is measured in the following way. We say the task is to explore the environment. Hence, visiting a lot of different places in the environment during an eval- uation is rewarded. One way of implementing this is to put a virtual grid over the whole arena (e.g., many little squares of side length 0.01 over the unit square) and keeping track of how many different grid cells were visited by the robot. The fitness is this number of visited cells. Do a number of independent runs, implement some output to plot trajectories that are generated by evolved be- haviors, and interpret your results.



For task 1 please turn in:

- please zip your submission in a single file named:

  `evoRobo_sheet2_YOURLASTNAME1_YOURLASTNAME2.zip`

- a readme file with the full names, a list of the tasks and subtasks you have completed and/or a list of tasks/subtasks you have not completed
- accepted file formats for plots: png and jpg
- all your code (either in C / C++ or python)
- full output of a typical run (for each generation: num. of generation and current string of that generation)
- Provide a short text as answer to b). You can add a plot if you like but that's not required.
- Provide a mathematical estimation of the expected number of generations along with a short explaining text and a comparison to your empirically obtained result from simulations.
- optional but very preferable: include a video where you go through and explain your code.

For task 2 please turn in:

- all your code (either in C / C++ or python)
- Plot a typical trajectory of your robot for the best individual (controller) of a run.
- Write a short text whether you are satisfied with your results and what would be possibilities of improving them.
- optional but very preferable: include a video where you go through and explain your code.