MICRO-515: Evolutionary Robotics
Prof. Dario Floreano

Exercise 1: Brain Controller Evolution in RoboGen

Jan Petrs (jan.petrs@epfl.ch)
Alexander Dittrich (alexander.dittrich@epfl.ch)
Juliette Hars (juliette.hars@epfl.ch)

For evolving robot bodies and controller, we will use in the following exercises RoboGen (http://robogen.org/app/). RoboGen[1] is an open-source educational platform for co-evolution of brains and bodies developed at the Laboratory of Intelligent Systems. It provides a basic set of evolutionary algorithms in the evolution engine, a physics simulation engine for evaluating the evolved robot and further utilities for generating 3D printable files of evolved bodies and deploying evolved neural networks controllers to embedded hardware.

RoboGen comes with extensive documentation, which you should frequently refer for solving these exercises (https://robogen.org/exercises/). You can use RoboGen via a Graphical User Interface, command line or a convenient web platform. We strongly recommend to use "RoboGen on the web" for these exercises (http://robogen.org/docs/robogen-on-the-web/).
See https://github.com/lis-epfl/robogen if you wish to look at the code for RoboGen.

Register your team

The final (graded) RoboGen project (Exercise Sheet 6) will be group based. You should form your groups today on Moodle (RoboGen project groups Group choice). The maximum number of groups is 10. You can also do Exercise Sheets 2-5 in your groups if you wish, but it is not mandatory.

Goal

The objective of Exercise Sheet 1 for you is to familiarize yourself with the RoboGen Evolutionary Robotics platform. This exercise sheet will focus on the evolution of the brain (controller) only. Future exercise sheets will include co-adaptation of the body and the brain. The evolved controller will drive a simple differential wheel drive cart robot (similar to an e-puck) that has to navigate in an environment as fast as possible while avoiding obstacles.

[1] Auerbach, J. E., Concordel, A., Kornatowski, P. M., & Floreano, D. (2018). Inquiry-based learning with RoboGen: An open-source software and hardware platform for robotics and artificial intelligence. *IEEE Transactions on Learning Technologies*, *12*(3), 356-369.

Learning objectives for this laboratory

- Learn how to perform a brain evolution in RoboGen.
- Basic fitness function design and implementation.
- How to test the generalizability of the evolved solutions.
- How to deploy the evolved brain to a real robot

Theory

Neural Controller

The robot is controlled by a neural network which transforms the inputs from multiple infrared (IR) sensors into motor commands for the left and right wheels of the robot. The neural network has 4 input neurons and 2 output neurons for the 4 IR sensors and 2 wheels, respectively. Inputs are scaled to fit the range [0,1] to improve the speed and stability of the learning process. The nonlinear activation function is the sigmoid function, constraining the output between 0 and 1.

Genetic Algorithm

The Genetic Algorithm evolves the synaptic weights of the described neural controller. The set of synaptic weights, each coded using floating point values, are therefore the genome of each individual in the population.

After randomly initializing a population of genomes, the population is evolved using **tournament selection**, **one-point crossover, mutation** and either ***"mu+lambda"*** **("plus") or "*mu, lambda*" ("comma") replacement**. The genomes of the first generation are initialized randomly with each weight in the range [−3,3]. Every individual is evaluated in each generation using the fitness function that you will design in *obstacleAvoidance.js* (more on this below).

The parameters that you can configure:

- choose "*plus*" or *"comma"* replacement,
- the values for *mu* and *lambda*,
- the tournament size,
- the crossover and mutation probabilities, and
- the mutation standard deviation.

As you saw in the programming exercises, the convergence rate, and even the success of the algorithm, can rely heavily on the selected parameters.

Getting Started

Go to the RoboGen web app: http://robogen.org/app/.

> **Note:** If you receive a "404 Error" ensure you access via "http://..." and not "https://...", as some browsers access URLs by default with secure HTTP.

We are going to evolve a controller for the cart robot in this exercise, so familiarize yourself with the RoboGen visualizer by clicking on the pre-evolved cart robot demo on the home tab of the web app.

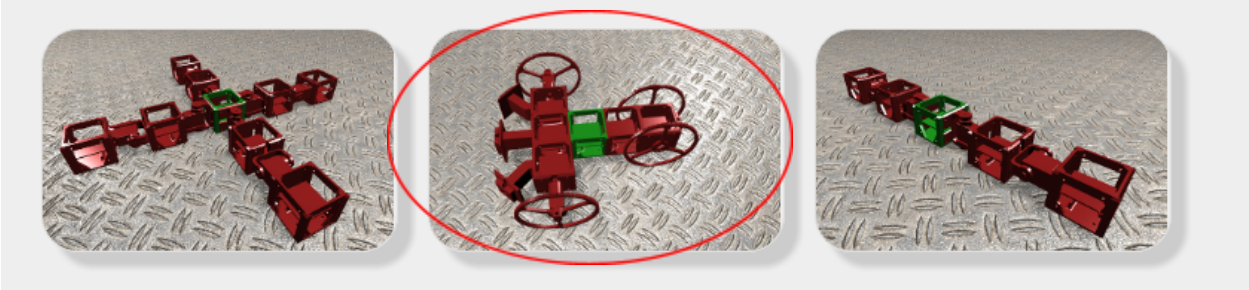


Figure 1: We will evolve a controller for the cart robot in this exercise sheet.

> **Note:** Make sure that you close the dynamic visualization tab when you are done to free up resources.

Simulations, Evolutions and Exercise files

To perform simulations and evolutions, go to

- ***Advanced*** tab of the web app
- ***RoboGen 2022*** folder
- ***es1*** folder

This folder currently contains an empty "`robot.txt`" file. You should download "`Exercise1.zip`" from Moodle and upload the contents to the "es1" folder on the RoboGen app.

- In the **es1** folder, click the "**Upload**" button at the top of the screen

The folder contains the following files:

- `evolConf.txt`
- `obstacleAvoidance.js`
- `robot.txt`

- `simConf.txt`
- `startPos.txt`
- `arena1.txt`
- `arena2.txt`

How to run a simulation in RoboGen?

For simulating (not evolving) the robot, you just need the following files:

1. `robot.txt` – the robot description file describing the physical geometry of the robot
2. `simConf.txt` – the simulation configuration file describing the physical parameters of the environment. This file inherits two other files
    a. `arena1.txt` - defines the obstacles in the arena
    b. `obstacleAvoidance.js` - JavaScript file where you should implement the fitness function

    These are assigned to the two parameters in `simConf.txt` (obstaclesConfigFile = `arena1.txt` and scenario = `obstacleAvoidance.js`)

See https://robogen.org/docs/evolution-configuration/#Simulator_settings for more details about the `simConf.txt` file.

To start your simulation, click on the “Start a simulation” button.

In the “Start a new simulation” box that appears, type your **working directory**:

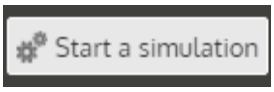
- /localStorage/Robogen2022/es1/

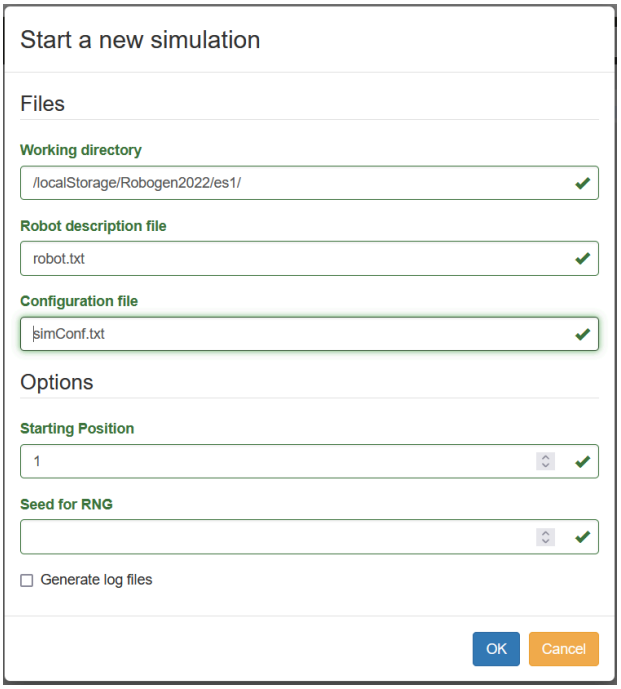
The **robot description file**:

➔ `robot.txt`

The **configuration file:**

➔ `simConf.txt`





The box will turn green if the path and filenames are valid.

Use **“Starting position”** if you have set startPositionConfigFile = `startPos.txt` in `simCong.txt` and `startPos.txt` contains more than one possible starting position, you can select which to use (line number).

“**Seed for RNG**” is the number that acts as a seed for generating random numbers. To randomly assign a seed, you can leave it blank. This seed is useful when you want to regenerate the exact same simulation that you have generated in the past.

Right now, leave “**Generate log files**” unselected. When you have evolved a good controller, you should select “**Generate log files**” and provide a name in “**Output directory for logs**” for a place to save the `NeuralNetwork.h` file generated by the simulation. When you build a physical robot in the future, you will need to download the `NeuralNetwork.h` file so that it can be used to program the microcontroller of the real robots to make it move.

After typing these file names, you can click OK to visualize your robot. You should see that the robot does not move. This is because the robot doesn’t have a brain yet!

How to speed up the evaluation of each population in RoboGen

There are two ways to do this. The first is to use **parallel processing** and the second is to use **distributed computing**.

By using **parallel processing**, you can speed up the computation by taking the advantage of your multicore processor. To use it, navigate to the "Computation Tasks" and increase the "Maximum number of parallel fitness evaluations threads" by moving the slider.

> **Note:** this number you select should be less than the number of cores you have available.

For **distributed computing**:

1. Decide on a group name, e.g. group_4.
2. In the "Computation Tasks" tab, go to the text box called "Group name" and enter the group name.
3. Click "Join".
4. Make sure your group name is selected.
5. Each group member should select the "Maximum number of parallel fitness evaluations threads" using the slider.
6. One member of the group can now run the evolution in the Advanced tab as usual.

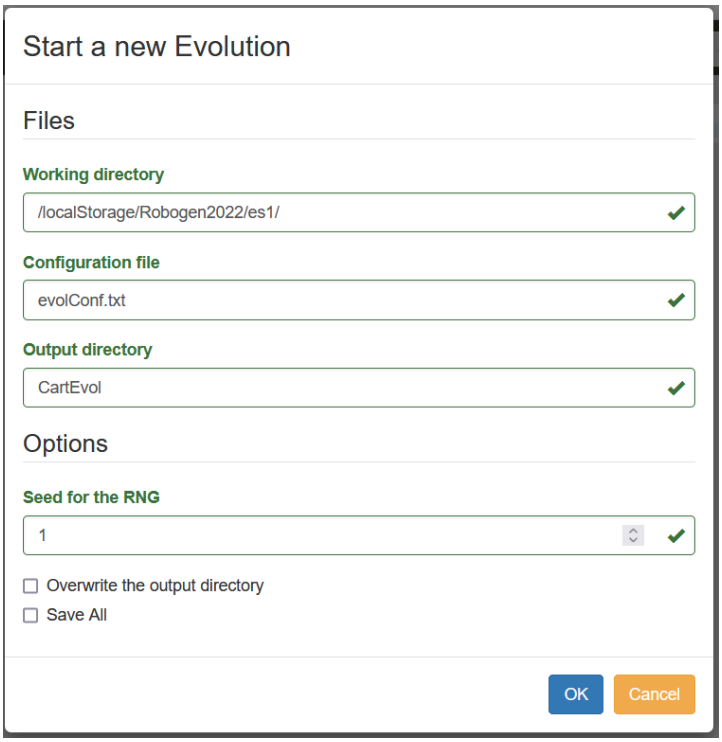
How to run an evolution?

Like the simulation, evolution needs a configuration file with additional parameters for evolution.

- `evolConf.txt` - contains the evolution parameters. This file also contains the links to
  - `simConf.txt`
  - `robot.txt`

Click OK to start an evolution.

Tip: if the software is having some problem, try refreshing the page.



Warning all data is being saved to a virtual file system within your web browser. If you want to save anything for later, download it to your home directory!

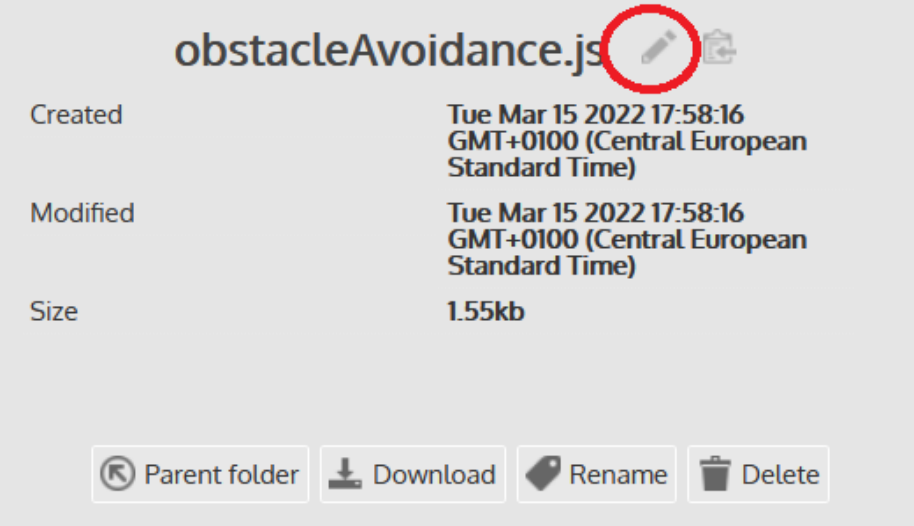
Fitness Function (aka Scenario Definition)

Scenarios in RoboGen can be defined by short pieces of ECMAScript/JavaScript. Please read through the accompanying document "Writing_a_RoboGen_Scenario.pdf" for complete details. As mentioned previously, the objective of this TP is to evolve a neural controller for a simple differential wheel drive cart robot that has to navigate in an environment as fast as possible while avoiding obstacles.

Exercise 1.1: Fitness function

Design and implement a fitness function that would allow the robot to navigate in the arena as fast as possible and without touching any walls. To do this, modify the code in `es1/obstacleAvoidance.js`.

> **Note:** you can do this on the RoboGen app by clicking on `obstacleAvoidance.js` and then clicking on the "edit file" pen symbol.



Read `obstacleAvoidance.js` carefully. You should see that information about the robot's behavior is collected after each simulation step in `afterSimulationStep` and that, at the end of the simulation, the fitness is computed in `endSimulation`.

In `obstacleAvoidance.js`, the fitness function is highest when the robot moves as fast as possible:

```
setupSimulation: function() {
      Sets starting position;
},
afterSimulationStep: function() {
      Calculate mean velocity;
      Calculate delta velocity;
      Calculate maximum IR reading at this time step;
}
endSimulation: function() {
      return mean velocity during simulation;
      // your fitness function goes here.
}
```

Try the first evolution without modifying anything, and then try to implement your own fitness function.

For inspiration, see how the two fitness functions included in RoboGen are written:

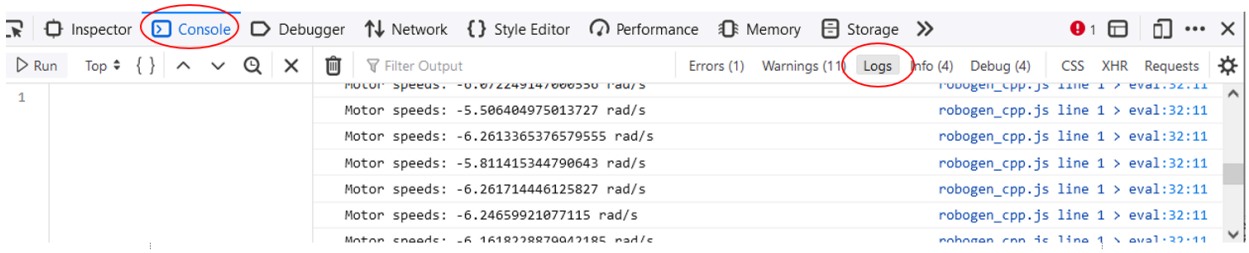
1. `chasing_scenario.js` – get as close as possible to a light source: https://github.com/lis-epfl/robogen/blob/8b710b93221882cdb9b970f55bf84d287dc2e4be/examples/chasing_scenario.js
2. `racing_scenario.js` – maximize the distance from the start position: https://github.com/lis-epfl/robogen/blob/8b710b93221882cdb9b970f55bf84d287dc2e4be/examples/racing_scenario.js

To get you started, we have provided `maxIrVals` and `minDistance` in `obstacleAvoidance.js`. See the end of "Writing_a_RoboGen_Scenario.pdf" or https://robogen.org/docs/custom-scenarios/ for more methods you can use to get information about the motors, sensors, position, orientation, etc.

To see what values you are getting from these methods, first add `console.log()` to your .js file, e.g.

```
var motors = this.getRobot().getMotors(); // array, motors[0] is
first motor, motor[1] is second, etc.
var motorOne = motors[0].getVelocity(); // velocity of first motor
[rad/s]
console.log("Motor speeds: " + motorOne + " rad/s"); // print
velocity
```

To see where the value of `motorOne` is printed in Firefox, use “Ctrl + Shift + K” (Windows) to see the console log and select Logs on the top right.



Finally, `getFitness` will return the minimum fitness across all simulations. This happens if we have more than one simulation per individual per generation (e.g. if we have multiple starting positions). So, if we evaluate a robot in multiple simulations, it is only as good as its worst case (e.g. the worst performing starting position).

Before trying to evolve with a given fitness function, make sure it works by just running the simulator.

Remember that GA is sensitive to the selected GA parameters.

After the evolution is finished, you can download the `BestAvgStd.txt` file from the results directory and run the provided `plot_results.py` file to create a fitness graph.

Questions:

- Did you get the desired behavior with your fitness function?
- What strategies did you observe during the evolution, and why were they good/bad in terms of fitness?
- What is the most implicit fitness function you can find?

Exercise 1.2: Generalization

We will now test the best individual obtained through evolution. To do so, go to the Advanced tab and modify `es1/simConf.txt`.

You can increase the time your robot is being tested by changing the `simulationTime` parameter. Do not change the other parameters (especially `timeStep`). Save this file and then choose “Start a Simulation”. Change the robot description file to your best individual, i.e. `my-experiment-path/GenerationBest-N.json`, where N is the final generation, and run.

You can also change the starting position of the robot by modifying `es1/startPos.txt`

- https://robogen.org/docs/evolution-configuration/#Starting_position_configuration_file

and add obstacles by modifying `es1/arena1.txt`

- https://robogen.org/docs/evolution-configuration/#Obstacles_configuration_file.

Additionally, you could also try the simulation with different arenas by changing the arena used in `simConf.txt` to `arena2.txt`.

Questions:

1. If you increase the simulation time, does the robot continue to perform well?
2. When you move your robot to a different start position, does it still work?
3. When you add obstacles in the environment, does your controller still work?

    - If your controller didn't generalize to these three tests, what could you do to fix the problem?
      Try to evolve a robot able to perform well in all previous conditions (increased simulation time, different starting positions and different obstacles positions).

Exercise 1.3: Deployment file to real system

See if your robot can solve the maze in arena2.txt using different starting positions. If not, try to do more evolutions (by changing parameters) to solve the maze problem.

We will need the `NeuralNetwork.h` file of your robot. To generate this file:

1. Click "Start a simulation".
2. In the popup window, select "Generate log files" and provide a directory in "Output directory for logs".
3. Navigate to your chosen directory (you might need to reload the RoboGen app for it to be visible) and download
    a. `NeuralNetwork.h`
    b. `obstacleAvoidance.js`