Evolutionary Robotics Laboratory

Exercise Sheet 1: Evolving a controller in RoboGen

Euan Judd (euan.judd@epfl.ch)
Krishna Manaswi Digumarti (krishna.digumarti@epfl.ch)

Goal. The goal of this laboratory is for you to become familiar with the RoboGen Evolutionary Robotics platform. This exercise sheet will focus on the evolution of the brain (controller) only while all 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.

Learning objectives for this laboratory.

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

Assignments.

- Complete the instructions in this PDF during the laboratory.
- Every student will submit the best controller found by their evolution on Moodle (24 March 2022: “Exercise Sheet 1 controller submission Assignment”). Multiple submissions are possible. The three best robot controllers will be tested at the beginning of the next class in a 3D printed version of the robot that we have prepared for you.

Register your team

- You should make a team of 4 people. Register on this google spreadsheet: bit.ly/35UNwWC

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 so the output will always be between 0 and 1.

Genetic Algorithm.
A genetic algorithm is used to evolve 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 initialising 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).

Tournament selection is used after the entire generation is evaluated. In each tournament, a set of individuals is randomly selected from the population. The best individual from this set is then selected as a parent. The child from this parent can then either come from this parent alone or a pair of parents. If the latter option is selected (where each of the two options is chosen with a certain probability) then another tournament is used to select the second parent as well. The new child is a clone of the parent in the former while the child is a combination of the two parents in the latter using one-point crossover. The child is then mutated by adding a number, drawn from a Gaussian distribution with a mean of zero, to each of the genes (weights) with a certain probability.

Replacement is then used to form the next generation. In "plus" replacement, the next generation includes both the *mu* parents and *lambda* children. Elitism is also used when creating a new generation, this is where the top-ranking *mu* individuals from the previous generation are also copied to the new population without modification. This ensures that the best solutions from each generation are not lost due to mutation or crossover. On the other hand, the *mu* parents are discarded in *"*comma*"* replacement and only *mu* of the *lambda* children are copied to the new population (*lambda>=mu).* This replacement strategy favours exploration as the best individuals from the previous generation can be lost.

The parameters that you need to 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.

The degree to which each of these influences your evolutionary process will vary depending on the problem.

Getting Started. We will use the RoboGen software that will run directly in your web browser (http://robogen.org/app/). This offers the benefit of being available to use on any

computer without having to go through an installation procedure. Additionally, it supports distributed computing so that the computational cost of your evolutionary process can be shared with your teammates. (Note: You can install a desktop version of RoboGen as well. The user interface version (https://robogen.org/docs/robogen-user-interface/) was recently tested on Ubuntu 18.04 and works well but the other options (https://robogen.org/docs/get-started/) may not. However, you should use the web app for this laboratory.

To get started with the online version, visit http://robogen.org/app

Click on one of the robots on the *Home* tab of the web app (pictured above). This will run a demo simulation of the robot. Try these out first to familiarize yourself with the visualizer. Test the cart robot (centre in the picture above) as this is the robot that we are going to evolve a controller for today.

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

Simulations, evolutions and exercise files

Simulations and evolutions can be performed under the *Advanced* tab of the web app. Click on *Advanced* and then on *RoboGen 2022* to see a list of all exercises for this academic year. These folders currently contain an empty "robot.txt" file but should be populated by you at the beginning of every laboratory using the zipped folder that we will put on Moodle each week.

If you haven't already, download the "Exercise1.zip" folder from Moodle, unzip it and upload the contents to the "es1" folder on the RoboGen app. This is done by first clicking on "es1" and then the "Upload" button at the top of the screen. You do not need to reuploaded these files each time you open the webapp.

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?

You need two files to run a simulation in RoboGen:

1. a robot description file (e.g. robot.txt in the *exercise 1* folder) and
2. a configuration file (e.g. simConf.txt in the *exercise 1* folder).

The robot description file describes the physical geometry of the robot. The configuration file describes the physical parameters of the environment. The configuration file inherits two other files and assigns them to two parameters called obstaclesConfigFile and scenario. As the name suggests, the contents of the file assigned to obstaclesConfigFile defines the obstacles in the arena (e.g. obstaclesConfigFile = arena1.txt). The file assigned to scenario is the JavaScript file where you should implement the fitness function (e.g. scenario = obstacleAvoidance.js).

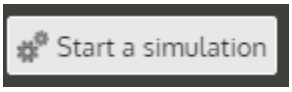
As we are going to work with the cart robot today, let's simulate the cart robot that you saw in the demo.

Robot description file: For this exercise sheet, you will be using the robot described in ***robot.txt*** file. You will be modifying or even creating the robot description file in later exercises.

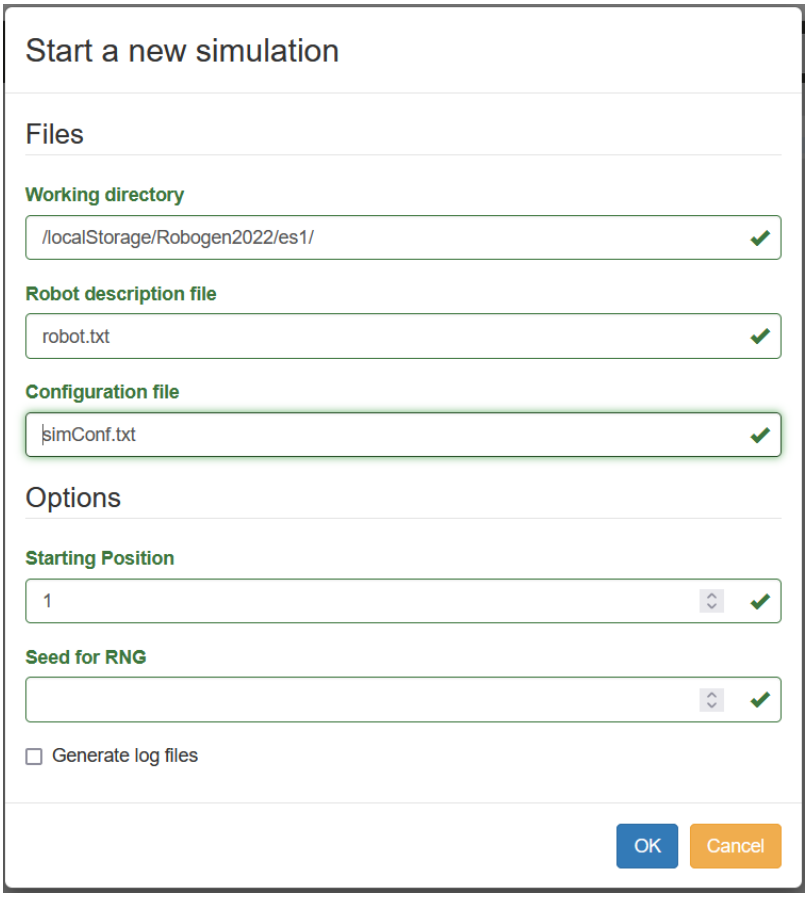
Configuration file: The configuration file for this exercise sheet is ***simConf.txt***. Have a quick look at this file now along with the files assigned to obstaclesConfigFile and scenario.

NOTE: All the details about the file, parameter and usage are available in the RoboGen webpage (e.g. see https://robogen.org/docs/evolution-configuration/#Simulator_settings for the simConf.txt file parameters).

Once you are ready and all the necessary files required are present in your folder, you can run a custom simulation. To start your simulation, click on the "Star a simulation" button.



In the “Start a new simulation” box that appears, type your working directory (e.g. the path to the exercise 1 folder in the webapp), robot description file and configuration file. You have to manually type the file names. The box will turn green if you type a valid file name and path.



In addition to these two files, the RoboGen simulator provides a few other options.

You can select which starting position to use in startPos.txt if it contains multiple options (one on each line) and you have specified startPositionConfigFile= startPos.txt in simConf.txt. This is done by writing the line number of the start position you want to use in startPos.txt under “Start Position”. Currently, there is only one start position in startPos.txt which is also the default value if you had not set startPositionConfigFile.

“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.

Output directory for logs is the folder where the NeuralNetwork.h file of the simulation will be generated. This file can be used to program the microcontroller of the real robots. Hence leave it blank until you have a best controller that can navigate fast and avoid all obstacles.

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 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 multi-core 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.

In addition to this, you can distribute your computation across various physical devices.

To distribute your computing navigate to the “Computation Tasks”. Type the name for your computation group and press join. After this, press the radio button next to your group name. You could ask all your peers to join this group to share your computation.

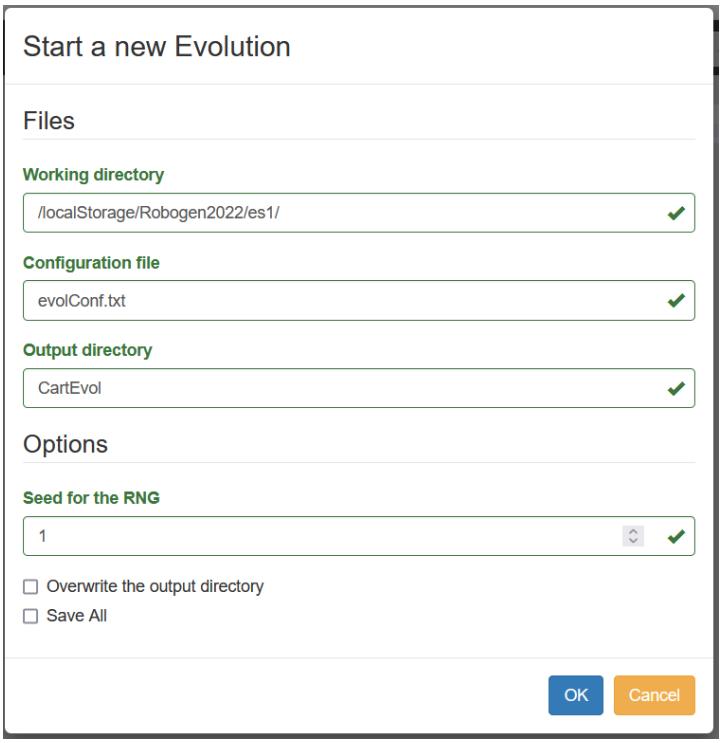
Warning When running the public group, the threads should be set to zero. Otherwise part of the computation could happen anyway locally slowing down the entire public users group.

How to run an evolution?

Like the simulation, evolution needs a configuration file with additional parameters for evolution. The evolution configuration file contains the evolution parameters (e.g evolConf.txt). This file also contains the links to the simulatorConfigFile (simConf.txt) and referenceRobotFile (robot.txt). For this TP, use ***evolConf.txt*** as the configuration file.

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!

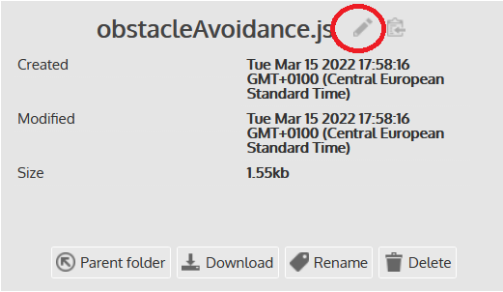
Fitness Function (aka Scenario Definition)

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 (similar to the e-puck) that has to navigate in an environment as fast as possible while avoiding obstacles.

Exercise 1 – Fitness functions

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 behaviour 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 the 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. To get you started, we have provided vectors of Velocity, deltaVelocity, maxIrVals, and minDistance that contains corresponding values at each simulation step of the simulation.

Finally, *getFitness* will return the min 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.

After the evolution is finished, you can download the *BestAvgStd.txt* file from the results

directory and run the provided *plot_results.py* file in python folder to create a fitness graph.

Questions:

- Did you get the desired behaviour 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 2 – Genetic Algorithm parameters

Re-run the experiment, each time changing one of these parameters in *es1/evolConf.txt*

- *mu, lambda, replacement, tournamentSize, pBrainMutate, brainSigma, pBrainCrossover*

Question:

For each evolution, download and have a look at the fitness graph. Do you see a difference with respect to your initial evolution?

Exercise 3 – 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 *nTimeSteps* 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* and add obstacles by modifying *es1/arena1.txt.* Additionally, you could also try the simulation with different arenas by changing the arena used in simConf.txt to *arena2.txt*.

For details on the contents of these files, see http://robogen.org/docs/evolution-configuration/#Simulator_settings

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). When you are satisfied with the results, submit your final robot using the following instructions.

Exercise 4 (only best robots) – Transfer to the real robot

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 show you how well some of your best solutions perform on a real robot next week. We will therefore 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 the *NeuralNetwork.h* file.
4. Upload it to Moodle “Exercise Sheet 1 controller submission”
5. If you have been selected, we will show your neural network on the real robot next week.