E1 Laboratory Exercises

Evolutionary Robotics

Davide Zappetti (davide.zappetti@epfl.ch)
Anand Bhaskaran (anand.bhaskaran@epfl.ch)

Goal. The goal of this laboratory is to start familiarizing with the Robogen Evolutionary Robotics platform by performing a brain only evolution of a robot. The evolved control 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 (for example, walls).

Learning objectives.

By the end of this laboratory you should have learned something about:

- How to perform a brain only evolution with the Robogen platform
- First steps in how to design a fitness function for evaluating a robotic task
- How to test whether or not the solution found through evolution can be generalized
- How to transfer the evolved brain in a real robot

Theory

Neural Controller. The robot is controlled by a neural network which transforms the sensory IR inputs received from the sensors into motor commands for the left and right wheels of the robot. Therefore, it has 4 input neurons and 2 output neurons. Inputs are scaled to fit the range [0,1] and the neuronal transfer function is chosen to be the logistic (sigmoid) function.

Genetic Algorithm. A genetic algorithm is used to evolve the synaptic weights of the described neural controller. The synaptic weights, which represent the genes of the individuals, are coded using floating point values. When put together, the genes form a genome.

A population of individuals is evolved, using tournament parent selection, one-point crossover, weight mutation and either *"mu+lambda"* or *"mu,lambda"* replacement. The genomes of the first generation are initialized randomly in the range [−3,3]. Each individual is evaluated based on the fitness function that you will design in `obstacleAvoidance.js` (more on this below).

In the case of "plus" replacement, the *mu* parents and *lambda* children are grouped together and ranked according to their measured fitness values, the top *mu* individuals are copied to the new population (elitism). This allows evolution to make sure that good solutions are not lost because of mutation or crossover. On the other hand, in the case of *"comma"* replacement, the *mu* parents are discarded, the *lambda* children are ranked, and the top *mu* of these are copied to the new population (*lambda>=mu).* This replacement strategy favors exploration.

After this step, a new set of *lambda* children are generated from the *mu* survivors with parents chosen by a tournament competition. With a certain probability two parents will be chosen (each by its own tournament) and one point crossover is applied to the pair, otherwise a single parent is chosen. In either the case the new offspring (either a clone or created by recombination) will have each of its genes mutated with a certain probability. In the case of mutation that gene will have a number drawn from a Gaussian distribution (w/ mean 0) added to it.

All of these parameters: choice of "*plus*" or *"comma"* replacement, the values for *mu* and *lambda*, the tournament size, the crossover probability, the mutation probability, and the mutation standard deviation are all configurable.

Installation

This year we will be using the new RoboGen software that runs on your computer. This offers the benefit of being available to use on any Linux computer without having to go through a complex installation procedure.

Step 1: Create a folder named **Exercise1 (without space)** in ~/Robogen/Exercises directory already present on the lab computer. To do this, you can open a terminal and execute the following command:

```
mkdir -p ~/Robogen/Exercises/Exercise1
```

Download the zip file provided in moodle named Exercise1 and extract it inside this folder. After this step, the folder structure should look similar to the diagram on the right.

```
Robogen/
└ Exercises/
   └ Exercise1/
      ├── cart.robot.txt
      ├── conf.sim.txt
      ├── conf.sim.txt
      ├── obstacleAvoidance.js
      ├── maze1.arena.txt
      └── maze2.arena.txt
```

```
chown -R :$(id -g) ~/Robogen/Exercises
```

Step 2: Download the latest release of Robogen app by using the following command

```
cd ~/Robogen

wget  https://github.com/lis-epfl/robogen-app/releases/download/V2.2/Robogen-2.2.0-x86_64.AppImage

chmod +x Robogen-2.2.0-x86_64.AppImage
```

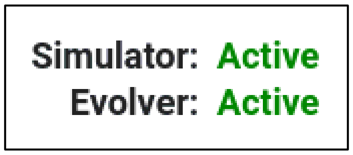
Step 3: To run the downloaded app just double click the app or execute the following command

```
./Robogen-2.2.0-x86_64.AppImage
```

If a popup to place the app on the system appears, press yes. From now on, you can start the app by just searching Robogen on the search bar.

By default, at the first usage, the software will download necessary files. You can see the downloading status at the top-right corner of the Robogen app window. When the app is ready to use, the status will change for both *Simulator* and *Evolver* to active.

Getting Started


Simulator: Active
Evolver: Active

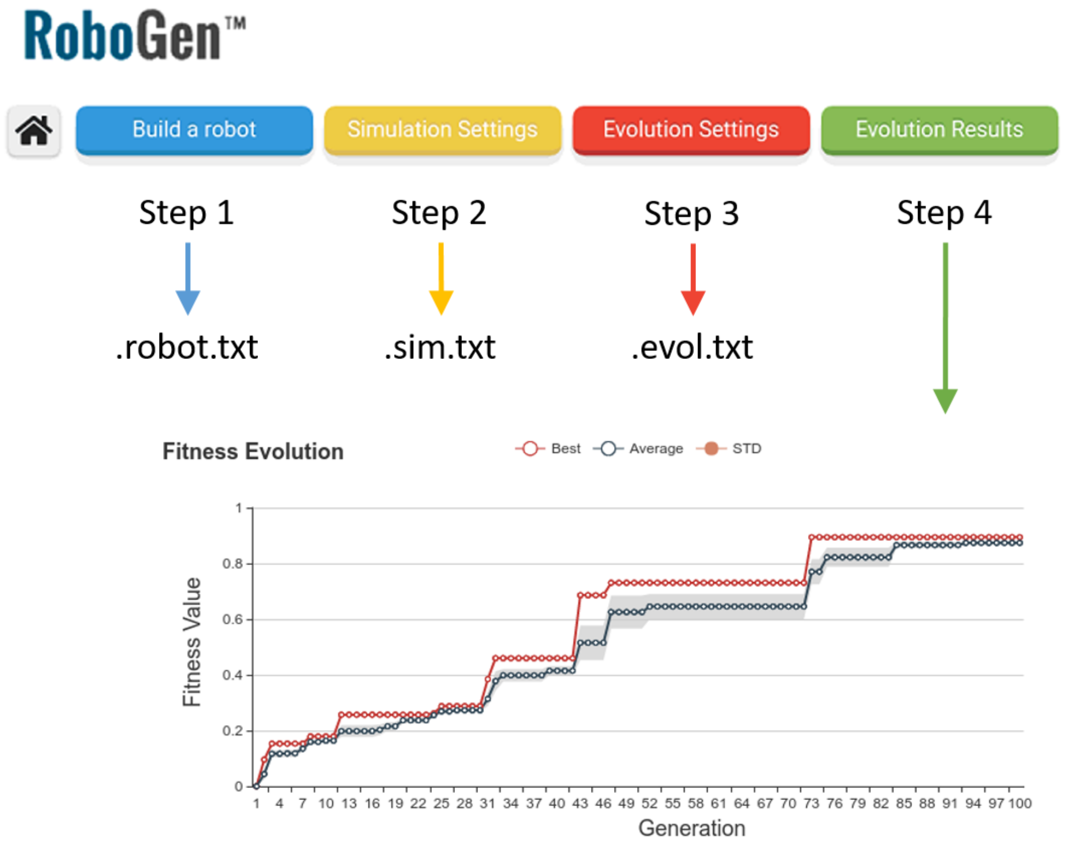
The app comprises two main components: simulator and evolver. As names suggest, the simulator is responsible of performing and displaying you a simulation of your robot, while the evolver is responsible of evolving your robot. At any time, you can see the status of the simulator and evolver on the top right corner of the app. In the case of inactive system, you will need to activate it by clicking a link appearing on the status. Clicking on it should activate the corresponding component.

Home page:

The home page contains two folder paths: Main Folder and Project Folder. The main folder path will automatically point to ~/Robogen/Exercises folder. Remember, only the files within this folder are available for the Simulator and the Evolver. The second folder is the project folder that contains all the project files. Go ahead and select ~/Robogen/examples/Excercise1 as your project folder. This step will automatically recognize the files and load it into the system.

At the bottom of the main page, the build number of the app is stated. It should be 2.2.0-beta. In case of any issue, do not forget to add build number in the email when contacting your teaching assistants.

Before we continue, let us examine the menu bar of Robogen App. The robogen app is organized to have four sections representing four steps that have to be done in sequence to perform an evolution (see image below).



1. Build a Robot:

This section allows to manually defining the body and brain of a robot or upload an existing robot body-brain file (e.g. name.robot.txt). For this exericise1, on the Robot file, cart.robot.txt would be already selected. Followed by this, you will have the following sections

- Robot Body: that describes the tree structure of the body
- Hidden neurons: that describes any hidden neuron in the brain of the robot along with their parameters
- Connection to describe the connection of the neural network and
- Bias that defines the offset

As you see the current robot has no hidden neurons and no connections. However, there is a slight bias that makes the robot moves.

Let' s now test the robot by clicking Test Me button on the right bottom of the application. Clicking this should open a new window. Click, drag and zoom to have a look on the robot. As you see, the robot has three IR sensors on the front and one IR sensor on the back. The robot also has front-drive mechanism, where the front two wheels are actively driven by DC motor while the back wheels are passive.

Click the button **P** on your computer to see the robot in action.

Wondering how the robot moves? Recap the importance of bias from the lectures.

The robot currently has a bias of 0.1 (clockwise) on the left wheel and 0.5 (anticlockwise) on the right wheel. Modify these numbers to get a robot that goes straight with its maximum speed.

2. Simulation Settings:

Now navigate to Simulation Settings tab. In this tab you will see that conf.sim.txt is automatically updated. Before we understand the settings, let's see the outcome. Go ahead and press ***Simulate Robot*** button on the right. Zoom in the simulation window and press P to simulate the robot.

Let's now examine which are the parameters of Simulations Settings.

- Robot File (The main Robot file from Step 1)
- Fitness Function File. (This is the important file that defines the good behavior of robot)
- Time step and simulation time are the parameters that defines how precise the simulation calculation should be and how long the execution should take place.
- Terrain friction and Gravity
- We can also have many other parameters for the simulation settings. Look here for more details.
    - obstaclesConfig file is what defines the obstacle (or maze) of the simulation. You can simply modify it by opening it in a text editor. On Robogen.org you can find a guide on how to modify and create obstacles.

Can you make the robot to escape from the arena1? Try playing with bias of the robot to make it escape.

3. Evolution Settings:

Evolution settings defines the parameters that you can change to set an evolution. This link will guide you through the detailed list of parameters.

As you see the evolution settings from conf.evol.txt would be automatically loaded into the User Interface. You should have the cart.robot.txt (from step 1) and conf.sim.txt (from step 2) as your robot and simulation file. This exercise focus mainly on Brain evolution, so keep the evolution mode to brain. You can play with other basic parameters (like number of generations, population and offspring size, Replacement Strategy, etc..) and Brain Variation Probabilities.

Note: Hidden neurons are not the topic of this exercise, but you are welcome to explore. However, in case you are adding hidden neurons, ***Add Oscillator Neuron Probability*** should be set to 0 as the DC motors cannot accept any oscillator neurons.

With default parameters, let's run our first evolution by clicking Evolve Robot button on the right bottom corner. Clicking Evolve Robot button leads automatically to section 4 where results of the evolution are displayed.

4. Evolution Results:

On this page you have two tabs, Past Evolutions and Ongoing Evolution. Past Evolutions are used to analyze your previous evolutions and Ongoing Evolution (where you are now), will show the status of your current evolution. However, both of these tabs have almost similar layouts.

The top part of the page contains the Fitness Evaluation Graph that shows the Generation on the X-Axis and the fitness value in the Y-Axis. Get back to theory to know the importance of this graph.

Below this graph, you have a table that shows the best, mean and standard deviations of the individuals in each generation. On the right, it is displayed the performance distribution within population. All individuals of that generation are represented with a square of different color according to the individual fitness value. By clicking on any individual of the generation is possibly to play the simulation of the corresponding individual.

What does the Fitness value mean??

Fitness value is the value that defines the performance of the robot. This value comes from the fitness function file that you used in Step 2. By default, the fitness function we have provided try to maximize speed of the robot regardless of obstacles.

Can Evolutionary Algorithm eventually solve the maze problem with this fitness function??

Yes, eventually, the well-tuned evolutionary algorithm can solve the maze just by evolving a robot that follows always the same trajectory. This is called overfitting problem. The ideal fitness function should generalize the problem in such a way that the same robot should be capable of solving the task irrespective of the maze configuration and of a certain trajectory.

How to ensure that your robot do not over fit a fixed trajectory?

Starting Positions: You can have multiple starting positions configured through the startPositionConfigFile in the Simulation Settings. This method will evaluate every individual (during evolution) starting from all the available starting points and the lowest value across them is returned as the fitness value of the robot.

How to test the robot in different arenas?

In simulation settings section of the robogen app it is possible to change the arena by changing: obstaclesConfigFile= nameofyourfile.arena.txt

By default we provide two arenas:

- maze1.arena.txt
- maze2.arena.txt

IMPORTANT:

- Always ensure to backup your files in a safe location.
- In case if you are moving the evol_results folder, ensure to move all the folders together.
- Ensure all the file that you use DONOT have any space in their file name

Next Step:

The goal of this session is to explore the simulation parameters, fitness function and evolutionary parameter. Once your robot is capable of solving the maze2, we could do hardware testing with the real robot.

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" for complete details. As mention in the goal, 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 (for example, walls).

Task 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 so, modify the code in Exercise1/obstacleAvoidance.js.

Reading carefully the file (obstacleAvoidance.js) You can realize that after each simulation step we collect various information about the robot's behavior in *afterSimulationStep*, and then at the end of the simulation the fitness for that simulation is computed in *endSimulation*.

In the file that we provide (obstacleAvoidance.js) the fitness function returns the mean velocity of the robot during the simulation.

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

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

Hint:

- For an efficient obstacle avoidance robot, the difference of velocities between the wheels is important
- Higher value of maxIrVals should be avoided (an obstacle is faced)
- Try the first evolution without modifying anything and then implement the desired fitness function.

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 position). So, if we evaluate a robot in multiple simulations it is only as good as its worst case and we will use its lower simulation fitness (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. If you see a value that makes sense then you are good to move onto evolution.

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?

Task 2 – Genetic Algorithm parameters

Re-run the experiment, each time changing a single one of these parameters in

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

Question:

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

Task 3 – Generalization

We will now test the best individual obtained through evolution. To do so, just click on the best robot of last generation. You could press P to Play/Pause the simulation.

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 by your results submit your final robot following the next instructions.

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

In the Simulation Settings modify the obstacles configuration to maze2.arena.txt. Try to have the robot in the starting position of (0.25, 1.25, 0.0) and check if your robot can solve the maze. If not, try to do more evolutions (by changing parameters) to solve the maze problem. Once you are done contact teaching assistants and we will help you to transfer your robot’s brain into the pre-built robot.