Evolutionary Robotics
Prof. Dario Floreano

E2 Laboratory Exercises

Body encoding & Simulation/Evolutionary parameters

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

Goal.

To further familiarize yourself with the RoboGen software suite by hand-designing a robot body and performing a brain only evolution of its controller. The designed-evolved robot has to navigate in a rough environment as fast as possible overcoming obstacles (for examples, pebbles and small hills)

Learning objectives

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

- Gaining an understanding of the tree structure used for encoding robot morphologies
- Learning to write your own robot description file
- Learning about how to set and use oscillatory neurons
- A more in depth knowledge of customizable simulation and evolutionary parameters
- Gaining experience evolving controllers for hand-designed robots

Getting Started: Similar to the Evolutionary Robotics exercises from last time, we will be using the new new RoboGen software that runs on your computer. If you did not install it yet, you can find instruction in the last week exercise 1 Manual.

Important:

- While last week you evolved controllers for a wheeled robot (the e-puck like robot), and this functionality does exist in RoboGen, for the rest of the semester **you will not be allowed to use wheels**. For this year's class the available parts are ***CoreComponent, FixedBrick, ParametricJoint, PassiveHinge, ActiveHinge, LightSensor,*** *and* ***IrSensor*** (if you specify addBodyPart=All when evolving morphologies, these will be the parts that your robots will be composed of).

Step 1.0

Last time you saw how to simulate a robot and how to define a scenario in order to evolve a controller for a pre-provided robot morphology. Now, it is time to understand how robots are defined in RoboGen.

Step 1.1: Create a folder named Exercise2 (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/Exercise2
```

Download the zip file provided in moodle named Exercise2 and extract it inside this folder.

In **"**Build a Robot" section, it is possible to manually define the body and brain of a robot or upload an existing robot body-brain file (e.g. name.robot.txt). For this exericise2, the Robot file, ***simpleRobot.robot.txt*** needs to be selected.

As already described in last class a Robot description file is composed of 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 its parameters
- Connections to describe the weights of the neural network
- Bias that defines the offset of the sigmoid neurons or the three parameters of an oscillatory neuron

Notes:

1. To observe and simulate simpleRobot.robot.txt you can press the ***Test Me*** button on the right.
2. For the details how robots are represented you should read through http://robogen.org/docs/guidelines-for-writing-a-robot-text-file/

Make sure you understand all lines of the robot description file ***simpleRobot.robot.txt***. Verify that you can answer the following questions:

- Which part of the file represents the robot's body and which part represents the robot's brain?
- For the body description, what does the indentation specify?
- What do the various numbers and strings on each line specify?

Step 1.2

Now speicify ***myRobot1.robot.txt*** as robot description file. This is a copy of simpleRobot.robot.txt that has had the brain specification removed.

Try adding/removing/modifying body components from this file and re-launching the simulation so that you can view your changes.

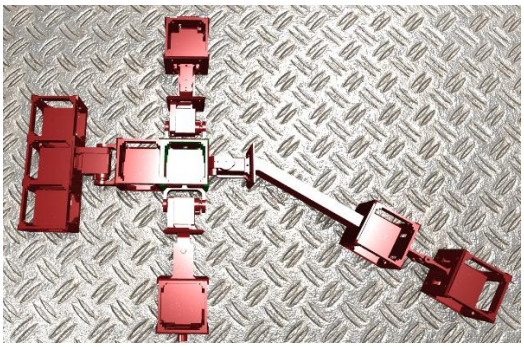
For example:

- What happens if you change ActiveHinge to PassiveHinge on line 2?
- What happens when you remove the last three lines of robot body?
- Can you add a light-sensor attached to one end of the robot?

Note: If your file is not properly formatted, you will receive an error message when you try to simulate this robot.

Step 1.3

Now, to make sure you understand how to write a robot description file, you should try to reproduce the robot shown below. To do this select ***Build your own robot*** in the robot file. You should pay careful attention to make sure all the components are included in the proper position, and all joint orientations match.



Step 2

Now that you understand how to modify a robot body, it is time to learn a bit about how robot brains are described.

For this exercise open ***myRobot2.robot.txt***. This is an exact replica of ***simpleRobot.robot.txt*** that you saw in action at the start of **Step 1.1**.

Now look at the Bias section of the robot, that were absent in ***myRobot1.robot.txt***. These lines define a very simple "brain" for your robot.

In general, brains in RoboGen are Artificial Neural Networks (ANNs). However, we also provide the ability to include some alternative types of artificial neurons, such as oscillators, in order to quickly find good locomotion abilities. Here, for example, we have specified that motor-neuron 0 of body part Hip1 is an oscillator with a period of 0.8s, a phase offset (relative to a central clock) of -0.8 periods, and an amplitude of 1 (meaning the joint's entire motor range).

Refer back to http://robogen.org/docs/guidelines-for-writing-a-robot-text-file/ for details of defining a robot brain (specifically the content under "The brain part").

Try modifying the parameters of the oscillators and see what happens. What happens if you increase the period? What happens if you decrease the amplitude (note amplitude must be in [0,1])? Or change the phase offset?

It is also possible to define a standard ANN in this description file, but it may be difficult to understand how the parameters influence behavior. Even so, in order to understand this you should try to define a simple neural network.

Replace each oscillator definition line with a line like

Hip1 0 Sigmoid 10 (replacing Hip1 with the appropriate body part name for each motor neuron). This will set each motor neuron to be a standard sigmoid, with bias 10.

- What happens if you simulate this robot?
- What happens if you change 10 to -10 on some of these lines?

Keep in mind that you have only set neuron biases, there are still no weights between neurons.

In order to manually set weights you will need to add weight lines to this file. For example remove all of the bias lines you just created and add this line to the file such that it comes **immediately after two blank lines** following the body description

Core 0 Hip1 0 10

This specifies that there is a connection with weight 10 from neuron 0 of the core component to neuron 0 of Hip1. Note that the core component contains the IMU sensor with accelerometer

and gyroscope. The indices of neurons there are as follows: *0,1,2* are the sensor neurons corresponding to *x,y,z* accelerations respectively. *3,4,5* are the sensor neurons corresponding to *x,y,z* gyroscope respectively. Neuron 0 of Hip1 corresponds to the motor neuron driving the servo motor of Hip1.

- What happens when you specify this weight?
- What happens if you also include a bias on Hip1 0? Remember to include a blank line after the weight description lines.

In general, it is not so intuitive to define the parameters of a neural network by hand, but do not worry we will be allowing evolution to define the weights for us!

Step 3

The last thing to understand about simulating robots in RoboGen is the simulator configuration file. Perhaps you modified this last time, but now we will look at it in a bit more depth. Go to the simulation settings tab and the file ***conf.sim.txt*** should be preloaded.

The full documentation for defining this parameters is available http://robogen.org/docs/evolution-configuration/#Simulator_settings

For now, it is important to notice the following parameters in this tab:

Robot File: This is the robot that will be simulated. **As you are using multiple robot files pay attention to which robot file the simulation uses.**

Fitness Function –specifies the scenario the robot should be evaluated in. There are two scenarios provided to you: "racing_scenario.js" and "chasing_scenario.js". racing scenario gives fitness based of the distance traveled by the robot and chasing scenario gives fitness based on how well the robot chases the light. Hoverer, chasing scenario are out of the scope of today's class. Alternatively, as you saw last time, you can define your own scenario with a small amount of JavaScript and provide the name of the js file . You should select the corresponding file in the Fitness function of the simulation.

obstaclesConfigFile – this optionally specifies the name of another file where environmental obstacles are defined. You should refer to http://robogen.org/docs/evolution-configuration/#Obstacles_configuration_file_ to see how to create obstacles. You can add obstaclesConfigFile=**maze.arena.txt**

startPositionConfigFile – this specifies the name of another file where starting positions and orientations are defined. The starting position configuration file contains 1 or more lines each containing the x-position, y-position and orientation of the robot (in degrees). Each robot will be simulated once for each starting position.

actuationFrequency (You cannot modify this) – this specifies the frequency (in Hz) that the controller operates at. Lower frequencies may lead to less jittery gaits, but may also be less reactive. 25 Hz is a reasonable value for the real hardware.

timeStep – this specifies the time step of the physics simulator (in s) or in other words, how much real time each iteration of the simulation represents. Larger time steps will allow running faster simulations, but can lead to unstable simulations. For the simulations that you want to be as close as reality use the value 0.005 in the time step, however for faster evolutions you can use any value between 0.005 to 0.04. NOTE: Inverse of actuation frequency should be a multiple of timestep. *nTimeSteps* – this specifies the length of a simulation in terms of number of time steps. The amount of real time represented by the simulation is *timeStep* * *nTimeSteps*

terrainLength and *terrainWidth* – these specify the dimensions of the terrain that the robot operates on (in meters). Note: that the ground is simulated as an infinite plane, so these are just used for rendering purposes.

terrainFriction – this specifies the coefficient of friction between the robot and the terrain

sensorNoiseLevel – this specifies the sensor noise level. Where the sensor noise is a Gaussian with std dev of *sensorNoiseLevel* * *actualValue*. i.e. the value given to the controller Neural Network is

*N(a * s)*

where *a* is actual, uncorrupted sensor value and *s* is *sensorNoiseLevel*

motorNoiseLevel – this specifies the motor noise level. Where the motor noise is uniform in range +/- *(motorNoiseLevel * actualValue)*

capAcceleration – this is a boolean flag that when true will return a very poor fitness for any individual whose acceleration exceeds caps (setable with *maxLinearAcceleration* and *maxAngularAcceleration* parameters). Setting this to true is useful for catching solutions that exploit flaws in the physics model and end up "flying" across the screen, mostly this arises when evolving morphologies, but may also be seen when just evolving controllers.

Try using different timeSteps and see how you can speed up the simulation, but be careful not to destabilize the simulator. Note that you should also modify *nTimeSteps* appropriately to keep a fixed overall simulation time. Also note that you cannot make the frequency of simulator updates faster than the actuation frequency, and in fact the period of actuation (inverse of frequency) must be a multiple of the timeStep!

Step 4

Now that you understand more about defining a robot and configuring the simulator, it's time to return to evolving controllers.

Navigate to the section Evolution settings. By default conf.evol.txt will be used as an evolutionary configuration file. By default, ***conf.evol.txt*** file will be used for evolutionary settings. Also make sure that you select the right robot file for your evolution. If you run multiple evolutionary experiments with the same output directory, the software will start creating directories:

evol_results, evol_results_1, evol_results_2, etc. **Keep this in mind when analyzing results. Make sure you are not looking in the wrong place.**

Additionally, remember that if you want to save results for future use, make sure to backup these folders.

Important: When you back up the folders, make sure to remove all the evol_result folders in the excersise2 directory. The software will be confused if random evol_result folder exists.

Finally, note the seed of the evolution for each step that you would like to replicate. The seed for the random number generator is important. If you are replicating an evolutionary experiment several times, you should be sure to specify different seeds for each replicate (this is important for making statistical comparisons!).

The evolution configuration file specifies parameters related to the evolutionary algorithm, as you saw last time, and also includes references to the simulator configuration file that will be used for the fitness evaluations, and to a robot file to evolve from (if applicable). All the possible evolutionary settings are documented: http://robogen.org/docs/evolution-configuration/#Evolution_client_settings

evolutionaryAlgorithm – this lets you switch between a basic evolutionary algorithm (*Basic*) or an indirect encoding known as HyperNEAT, which you will learn about later in the course. For this example we use HyperNEAT so that we can quickly evolve good gaits for our robot. The *neatParamsFile* lets you modify some internal parameters for HyperNEAT, but we will not touch this file now.

Step 5

Now, you will use everything you have learned from Steps 1-5 to hand-design a robot body and perform a brain only evolution of its controller. The designed-evolved robot has to navigate in a rough environment as fast as possible overcoming obstacles (for examples, pebbles and small hills).

- You can follow today's Step 1 instructions to design your robot body (write a robot description file in txt format).
- You can try to define the brain manually with what you learned in today's Step 2, or leave it empty and evolve it.
- You can simulate your robot in the example **maze.*arena.txt*** file to check its body-shape and eventually its hand-designed brain.
- The same **maze.*arena.txt*** can be used for performing evolution of the robot brain. However, you are free to define your own arenas and simulation configuration file to improve the evolution of the robot (set different obstacles and starting positions)

- The fitness function file in this lab can be set as “racing_scenario.js”, this will use automatically a scenario file with a fitness function that tries to maximize distance travelled in a certain time (and therefore robot speed). You are welcome to customize this scenario.