Evolutionary Robotics
Prof. Dario Floreano

Evolutionary Robotics Laboratory

Exercise Sheet 2: Body encoding & Simulation/Evolutionary parameters

Euan Judd (euan.judd@epfl.ch)
Krishna Manaswi Digumarti (krishna.digumarti@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 must navigate in a rough environment as fast as possible and overcome obstacles (for examples, pebbles and small hills)

Learning objectives

By the end of this laboratory, you should have:

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

Assignments

- Go through the provided PDF instruction file during the laboratory class
- Every team will submit the best evolved robot in txt format before next class.

Getting Started: Similar to the Evolutionary Robotics exercises from last time, we will be using the web-based version of RoboGen.

To get started, visit http://robogen.org/app

As you saw in the previous exercise sheet, you should click on the "Advanced" tab to be able to modify files and start custom evolutionary experiments or visualize your robots.

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

Important:

- Remember, all data is being saved to a virtual filesystem within your web browser. If you want to save anything for later use, be sure to download it to your home directory!

- While last week you evolved controllers for a wheeled robot (the e-puck like robot), for the rest of the semester (including the Robogen Grand Challenge) **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).

Exercise 1.1

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

You could download the files required for this exercise from Moodle and upload in the folder Robogen2022/es2/ folder in the Advanced tab. Now, click "Start a simulation", and enter the following

Working Directory	/localStorage/Robogen2022/es2
Robot description file	simpleRobot.txt
Configuration file	confBasic.txt

You should see a robot being simulated. Remember, **no evolution is occurring here**!

Now, close this tab and open simpleRobot.txt (either using the web editor, or by downloading and opening in the text editor of your choice).

Here you will see the robot description file for the robot you just simulated.

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 this information. 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?

Exercise 1.2

Close the text editor and now open myRobot1.txt. This is a copy of simpleRobot.txt that has had the brain specification removed, which will make it easier to try creating robot morphologies.

Try adding/removing/modifying body components from this file and re-launching the simulator

(now specifying myRobot1.txt as the Robot description file) so that you can view your changes.

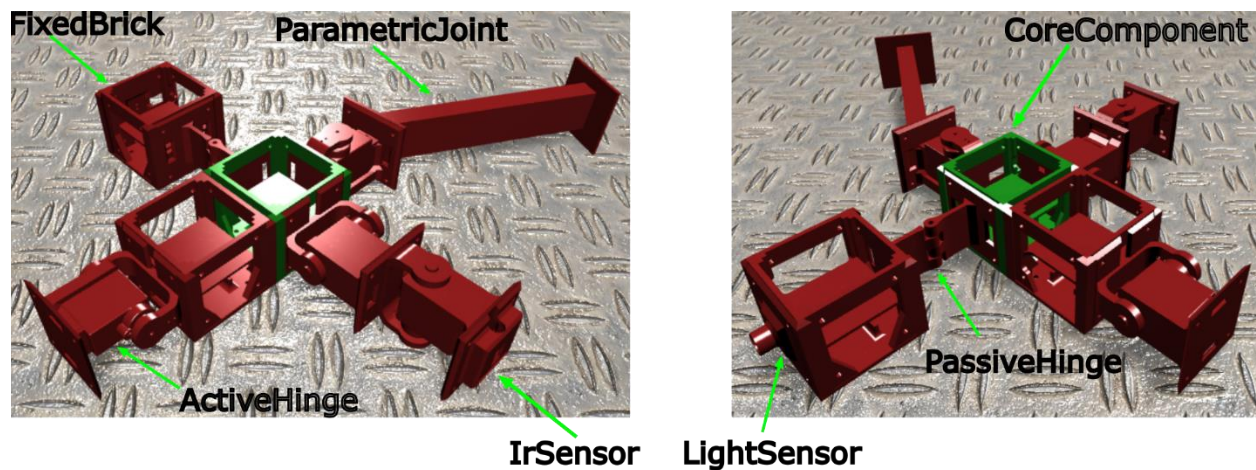
For example:

- What happens if you change ActiveHinge to PassiveHinge on line 2?
- What happens if you remove lines 6-9?
- 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.

Exercise 1.3

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



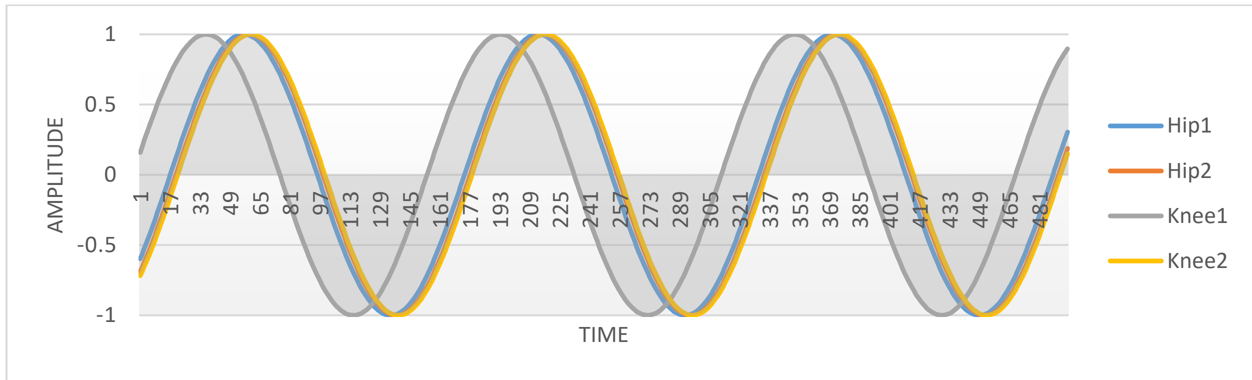
Exercise 2

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

From the Advanced tab open myRobot2.txt. This is an exact replica of simpleRobot.txt that you saw in action at the start of Exercise 1.

Now look at the lines from “Hip1 0 Oscillator 0.8 -0.8 1” and below that were absent in myRobot1.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, 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 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 behaviour. Even so, 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 and leave 3 empty lines after the robot description). 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.

To manually set weights, you will need to add weight lines to this file. For example, remove all the bias lines you just created and add this line to the file such that it comes **immediately after exactly 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? **Write the bias description after the weight descriptions with exactly one blank line between the last weight description and first bias description**.

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!

Exercise 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. Open myConf.txt

You are now looking at a simulator configuration file. The full documentation for defining this file is available http://robogen.org/docs/evolution-configuration/#Simulator_settings

Focus on the following properties in this file for now:

scenario – specifies the scenario the robot should be evaluated in. There are two scenarios provided to you: "racing_scenario.js" and "chasing_scenario.js". The racing scenario gives fitness based of the distance travelled by the robot and the chasing scenario gives fitness based on how well the robot chases the light. Hoverer, the chasing scenario is 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. We will not use obstacles in today's exercises, but you should refer to http://robogen.org/docs/evolution-configuration/#Obstacles_configuration_file_ to see how to create obstacles.

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.

timeStep – this specifies the time step of the physics simulator in seconds. In other words, how much real time each iteration of the simulation represents. Larger time step speeds up the simulations, but they can become unstable. Generally, you want to use as large of a time step as possible that maintains stability (this may depend on the complexity of the robot you are simulating). Use 0.005 for the most realistic simulations. For faster evaluations, you can use values between 0.005 and 0.04. NOTE: Inverse of actuation frequency should be a multiple of timestep.

actuationFrequency – 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.

simulationTime – Total length of simulation in seconds. Must be a multiple of timeStep. **One of nTimeSteps or simulationTime must be specified**.

terrainLength and *terrainWidth* – these specify the dimensions of the terrain that the robot operates on (in meters). Note: 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, a * s)*

where *a* is the actual, uncorrupted sensor value and *s* is the *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 the maximums (settable 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. This mostly 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. Also, note that you cannot make the frequency of the simulator updates faster than the actuation frequency and the period of actuation (inverse of frequency) must be a multiple of timeStep!

Exercise 4

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

From the advanced tab, click "Start an evolution"

Enter **myEvolConf.txt** as the Configuration file, **simpleExperiment** as the Output directory, and a number of your choice as the Seed for the RNG.

The Output directory specifies where to write results to. Unless you click "Overwrite" the evolver will not overwrite results. So, if you run multiple experiments with the same output directory it will start creating directories: simpleExperiment_1, simpleExperiment_2, etc. **Keep this in mind when analysing results. Make sure you are looking in the right place.**

Additionally, remember that if you want to save results for future use, be sure to download them to your home directory.

Finally, 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 (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). The possible options for this file are documented: http://robogen.org/docs/evolution-configuration/#Evolution_client_settings

Exercise 5

Now, you will use everything you have learned from exercise 1-4 to hand-design a robot body and perform a brain only evolution of its controller. The designed-evolved robot must navigate in a rough environment as fast as possible and overcome obstacles (for examples, pebbles and small hills).

Exercise 5 steps:

- You can follow today's exercise 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 ***arena.txt*** file to check its body-shape and eventually its hand-designed brain.
- The same ***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)
- You can set "scenario = racing" as the scenario file, this will automatically use 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 try with different scenarios. You could see the js file of racing scenario here.