MICRO-515: Evolutionary Robotics
Prof. Dario Floreano

RoboGen - Under the hood

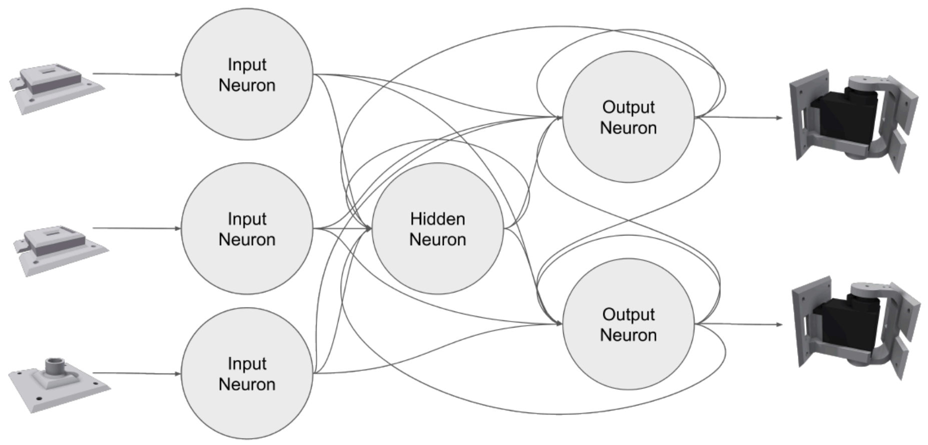
You might ask yourself, how RoboGen actually works behind the curtain of abstraction of the RoboGen web interface.

How do **neural network controllers** (brains) work in RoboGen?

In RoboGen, the behaviour of your robot is defined by the weights of Artificial Neural Networks (ANNs). Theoretically, large enough artificial neural networks can represent a large range of behaviours. The number of sensors and actuators of our robot define the number of input and output neurons. Each sensor signal is mapped to an input neuron, and each output neuron controls an actuator. Additionally, the neural network can possess a number of internal or hidden units not directly connected to any inputs or outputs. Simple neurons compute its output by first computing weighted sums of its inputs and then applying a sigmoid activation function. Alternatively, RoboGen can generate oscillator neurons, as they have been shown to drastically speed up the evolution of effective controllers for locomotion. Specifically, these oscillator neurons do not receive any input, but rather output a sinusoid oscillation as a function of time. Each oscillator neuron is defined by its period, its amplitude, and its phase-offset from a central clock.

Unlike the learning of weights of fully-connected deep neural networks with backpropagation, evolutionary algorithms enable the evolution of neural networks with multiple types of neurons and changing neural network architectures (e.g. variable numbers of hidden neurons).

Each neuron representation in this list stores the layer (input, output, or hidden) of the neuron, the type (sigmoid or oscillator) of the neuron, and its parameters (bias for sigmoid neurons; period, amplitude, and phase-offset for oscillator neurons).



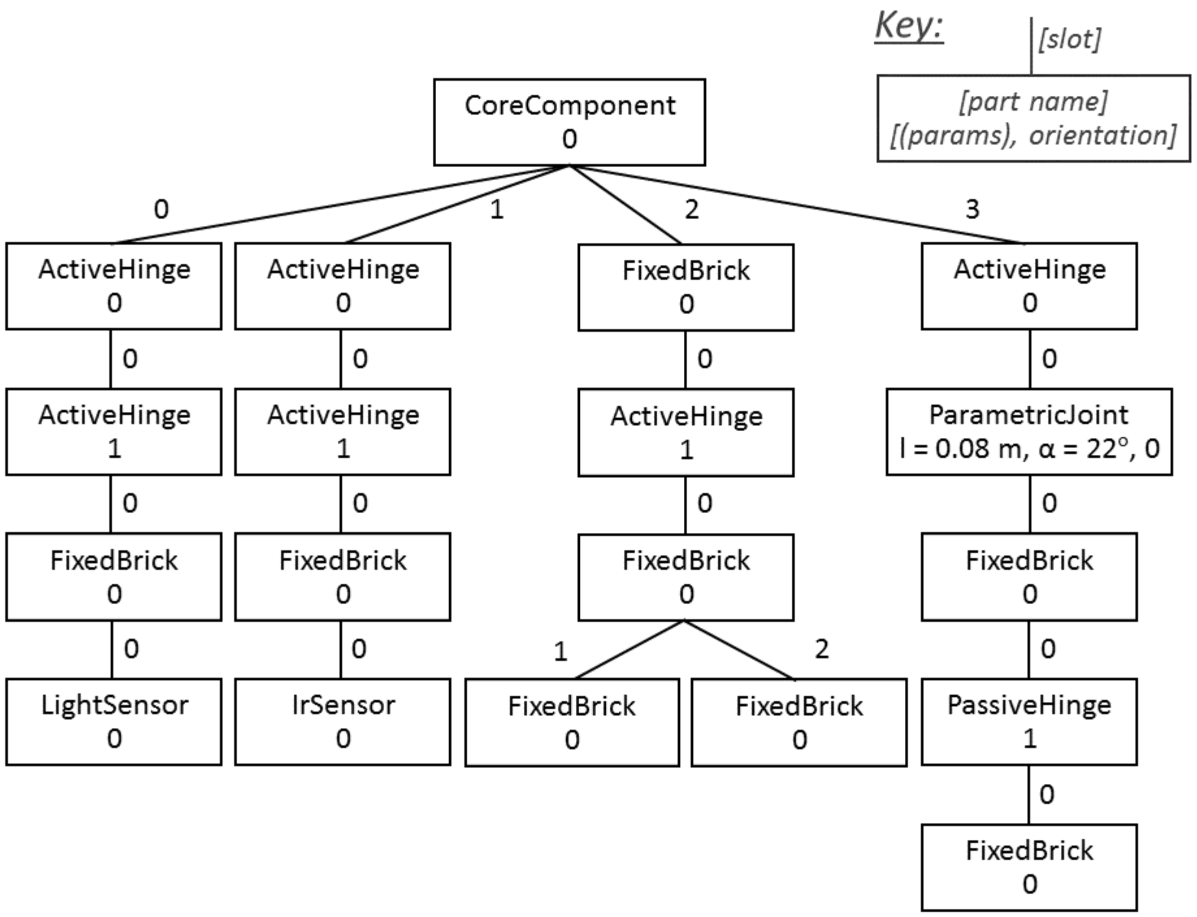
The connections and weights for the neural network are maintained in a single map: weights for the whole robot that maps ordered-pairs of neurons to floating point values. If an entry $weights[(i, j)]$Š exists, then the connection from neuron i to neuron j exists with weight $weights[(i, j)]$Š. If no such entry exists, then there is no connection from i to j in the neural network. For the purpose of applying evolutionary variation operators, this representation is converted into a single vector containing all weights and parameters.

How is **body morphology** defined in RoboGen?

RoboGen robots are made up of predefined and parameterized body parts, each of which is composed of one or more 3D-printed structural elements in possible combination with off-the-shelf electronic components: sensors, actuators, and an Arduino-based microcontroller. In order to support a large diversity of morphological possibilities, these parts can be assembled together into a variety of configurations.

The body parts are assembled together by means of plates (male) that slide into connection slots (female). Additionally, the system includes body parts with modifiable parameters (e.g., dimensions and angles), which allow for leveraging the customization capabilities offered by 3D printing. All RoboGen robots begin with the `CoreComponent`, which serves as the root of the tree. The other body parts fall into four categories: bricks, joints, sensors, and connectors. The bricks include the `CoreComponent` and `FixedBricks`: while the `CoreComponent` houses the microcontroller and battery, the `FixedBricks` are empty, but otherwise these parts are identical. Each `FixedBrick` measures 41x41 mm horizontally, is 35 mm high, and has (female) connection slots on its four sides. There are currently three different types of joints: `ParametricJoints`, `ActiveHinges`, and `PassiveHinges`.

`ParametricJoints` are rigid connections with a variable length and angle that allow the robot's basic shape to not be confined by a Cartesian grid. `PassiveHinges` include a passive joint with one degree of freedom composed of a brass axle for minimal friction. `ActiveHinges` include an active joint with one degree of freedom actuated by a servomotor. All joints contain two (male) connections plates at opposite ends. There are two types of sensor body parts: `LightSensors` and `IrSensors`, which can provide the robot with additional sensory feedback. The former are analogue photoresistors, which output a signal proportional to the perceived light intensity. They are directional to enable light following behaviour and calibrated to discount for ambient light.



How does the evolutionary algorithm change body and brain in RoboGen?

RoboGen comprises two main components: an evolution engine and a physics-based simulation engine. In short, the evolution engine runs an evolutionary algorithm by starting with a randomly initialized population of abstract robot representations, sending each representation to the simulation engine, which first translates each abstract representation into a concrete, physically simulated robot, simulates this robot in a virtual environment, and finally reports back a performance measure or fitness for the given robot. The evolution engine then eliminates the poor performing robot representations and reproduces the well performing ones (with variations). The newly created robot representations are next sent to be evaluated in the simulation engine, and the process repeats until a stopping condition is reached (usually a fixed number of iterations).

Evolution Engine:

The evolution engine contains all the components needed to run such an evolutionary algorithm on a population of robots: representation, variation, and selection mechanisms. The evolutionary algorithm is capable of operating in two modes: `brain`-only mode, which only evolves the controller or 'brain' of an existing robot morphology, and `full` mode, which evolves both the brain and 'body' (morphology) of robots. In brain-only mode, the body tree must be manually defined (or previously evolved). When using full evolution, it is possible to either "seed" evolution with an existing morphology or create the initial population completely at random.

At each generation, new body trees are created by mutating clones of well-performing robots using one or more of the mutation operators based on user-configurable probabilities. The body plan defines the inputs and output of the robot's brain, i.e., as sensor or motor body parts are added or removed from the body tree, input and output neurons (respectively) are added or removed from the brain. Additionally, hidden neurons may be added or removed based on configurable probabilities. When a neuron is removed, all connections to/from that neuron are also removed. When a neuron is inserted, zero-weight outgoing connections are created to all sigmoid hidden and output neurons. When a hidden or output neuron is inserted, it is chosen to be either a sigmoid or oscillator neuron based on a configurable probability. If it is a sigmoid neuron, zero-weight incoming connections are created from all neurons. In addition to these topological changes, the parameters of a robot's brain (connection weights and neuron parameters) are mutated by Gaussian perturbations based on user defined probabilities and magnitudes.

Sexual reproduction (AKA crossover) is possible when evolving ANNs with a fixed topology (e.g., doing brain-only evolution with 0 probability of adding additional hidden neurons), but otherwise is disabled for simplicity. When two brains are crossed over, the vectors of weights and parameters are concatenated into a single vector, single point crossover is applied, and this crossed-over vector is used to create a new brain representation prior to the application of the parametric mutations. At each generation, the current `mu` parents compete in tournaments (of configurable size) to decide which parents get to produce offspring. This process repeats until children are produced. After the children have been evaluated in the simulation engine the next set of `mu` parents are chosen by choosing the most fit of either the current children (`comma` selection) or the most fit out of the children and previous m parents (`plus` selection), and this process repeats for the user-specified number of generations.

In addition to the basic evolution mode just described, RoboGen also supports evolving brains via a more sophisticated EA, known as HyperNEAT, which indirectly encodes the weights and parameters of the neural network using an encoding inspired by developmental biology.

Simulation Engine:

Every newly created robot representation (brain and body) is sent to the physics-based simulation engine in order to evaluate its fitness. The abstract representation is translated into a physical model of the robot along with code for operating its brain, and this robot is placed inside a user-defined, simulated environment and allowed to act. The robot's fitness is computed either using one of the built-in fitness functions, or the user may write a custom fitness function in JavaScript.

By default, the environment is an infinite flat plane with Earth normal gravity. However, this can be modified in a number of ways, the most important of which is through the inclusion of obstacles. By defining a set of obstacles and light sources (either stationary or movable) that the robot can interact with, it is possible to create more complex environments such as rough terrains, mazes, stairs, goal locations etc.

Further information

Online documentation:	https://robogen.org/docs/robogen-software-suite/
Source code:	https://github.com/lis-epfl/robogen