Joint evolution of morphologies and controllers for realistic modular robots

D. Akrour[1], S. Cussat-Blanc[2], S. Sanchez[2], N. Djedi[1], and H. Luga[2]

[1]Biskra University, Algeria
[2]Toulouse University, France
[1]akrour_djouher@yahoo.fr

Abstract: Following Karl Sims seminal works, many approaches from the literature aims at generating artificial creatures using body-brain co-evolution. However, in simulation, creatures are not very realistic, they cannot be tested in physical robots. In this paper, we propose a system that can generate realistic walking artificial creatures. We co-evolve the morphology and the controller of virtual modular robots using GA. The morphology is generated by Graphtals while the global behavior of a creature is done by the cooperation of robot's modules moves. Each module has its own local controller, here based on an ANN. We integrate our system in Gazebo, a popular realistic robotic simulator. Experimental results show the capacity of our approach to generate realistic morphologies and behaviors with simulator parameters set up with realistic values. We expect the virtual robots generated with our system and trained in a realistic robotic simulator to better bridge the gap to reality.

Keywords: Artificial life, modular robots, evolutionary robotics, realism, gazebo

1 INTRODUCTION

Evolutionary robotics [16] is a wide research area that proposes to use evolutionary computation to generate robots' morphology blueprints and their associated behavior. Following Karl Sims seminal works [1, 2], many approaches from the literature aims at generating artificial creatures either in simulation [10, 13, 14, 15] or embedded in real world robots [8, 9]. In that way, body-brain co-evolution has been used to automatically generate autonomous adaptive robots [3].

However, in dynamic unpredictable or unknown environments, the more complex the task becomes, the more challenging the conception of the robot is. This is due to 1) the increasing of search space size when the robot morphologies and controllers are open to evolution and 2) the numerous possible interactions these robots can have with always more complex environments. When working in evolutionary robotics with adaptable morphologies, modular robotics [6] have shown to be effective [4]. Modular robots consist of a set of modules, either homogeneous or heterogeneous. If the tasks they must perform requires adaptation capacities, these robots can be self-reconfigurable [5]. Once evolved within a simulated environment, the robot body and its proper controller can be tested in real physical world. However, the behavior of simulated robots, as well as the simulated environment, needs to be as realistic as possible to minimize the reality gap which can make the obtained robot's behavior unstable.

The aim of our work is to evolve heterogeneous modular robots in the most realistic possible way. We propose a system that can evolve robots' morphologies along with their controllers. We integrate our system in Gazebo [11], a popular realistic robotic simulator often used to simulate large robots (humanoids, wheeled-robots, etc.). One of the key components of the simulation platform is the use of simulated sensors that produces a data stream which closely matches data from real-world physical sensors. The simulator provides for the environment and the simulated objects numerous physical attributes that we can set up with realistic values. Before evaluating robots in simulation, we designed a module interpenetration check routine to fix unsuitable creatures. To prevent the modules from going through each other in run time simulation, we modeled joints as modules.

The rest of the paper is structured as follows: In Section 2, we will present our virtual robots and explain how the morphology and the controller were generated together. Section 3 will briefly present the evolutionary process. In section 4, we will introduce the simulator and discuss the importance of the physical attributes and how they can affect the realism of the simulation. Section 5 describes our experiments with virtual modular robots and shows some examples of the evolved robots. Finally, section 6 concludes with a summary of this work and some possible future works.

2 VIRTUAL ROBOT

Robots' generation is inspired by Karl Sims simulated evolved creatures [1]. The robot's body is generated alongside with its controller. In this section we will present a description of our robots' morphologies and the process of creating the phenotypic virtual creatures from their genetic

representation. Then, we will describe the embedded distributed control system.

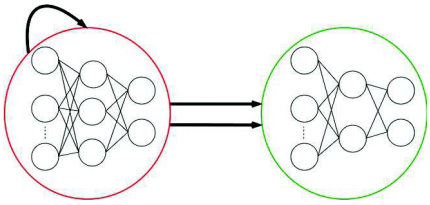

Fig. 1. A typical example of a genotype. The outer graph describes the morphology (assembly and features) while the inner graph on each node is the neural network controller.

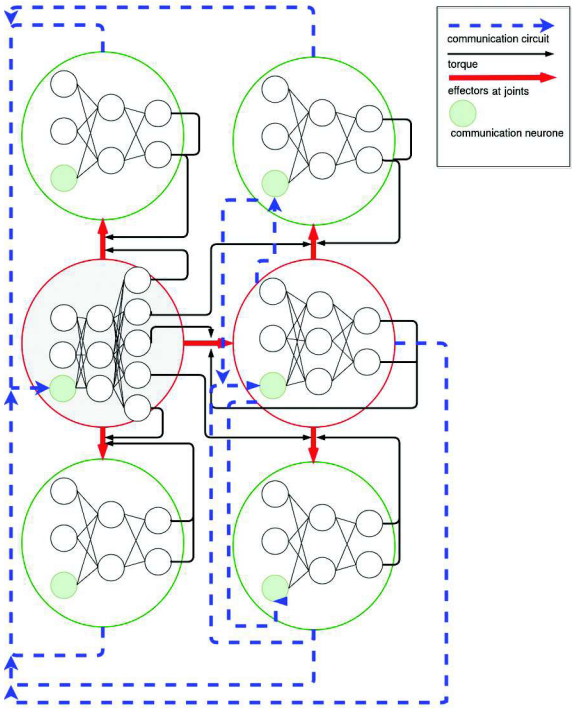

Fig. 2. The phenotype generated from the genotype shown in figure 1. Grey phenotypic node (module) is the root. It contains the global controller.

2.1 Robot morphology

Robots' morphologies are composed of a set of cuboid modules linked by joints of equal size. The fact that joints are modeled as modules without being invisible, will help to avoid the interpenetration between neighboring modules in runtime simulation. A module can have at maximum six joints which could be on the same face.

To generate diversified morphologies, we use *Graphtals* [1] as genetic representation. These are oriented graphs that consist of a set of nodes and connections. Each node of the graph contains the module phenotypic parameters (shape, size, available sensors, etc.) and a local controller that we will explain in the next section. Connections will determine the assembly plan of modules. Each one specifies the axis of rotation of the phenotypic joint and where it should be placed on the module face. Recursion and reflection in graphs allow duplication of nodes and, therefore, the possible emergence of modularity and symmetry in the generated morphologies. Recursive and reflective connections as well as the number of repetition for each application is specified by the genotype.

Firstly, we start generating the morphology from the root node in the graph. When created, new modules are assigned with a unique identifier and a parent. The root node has no parent. Recursion or reflection are applied on the fly. To increase search space, we propose two methods to apply recursion, we either repeat the module and all its children for all replications, or repeat the module itself and then add the children at the extremities. Concerning reflection, we replicate modules connected to the reflective connection. New replicated modules will be added in a predetermined face which could be in the same face. In this case, with only one root node, quadrupeds and hand-like morphologies are possible. Since we added a decreasing scaling factor, size of modules created from the same node may change. Of course these parameters are given by the genotype. Once done, we determine the position and the physical attributes of each module.

2.1.1 Morphology recovery

In some cases, unsuitable phenotypic robots may be generated. Modules can be interpenetrated. Therefore, before evaluating robots in the simulator, we designed a module interpenetration check routine to fix the unsuitable robots. This will let us recover morphologies that may be good. Following is a pseudo code of the check algorithm applied to the phenotype:

Algorithm 1 interpenetration checking

```
procedure Check (Module m)
  m. visited ← true;
  for children of module m do
    for 3 temptations do
      if interpenetration between m and all visited
      modules then
          In an order try one of these actions:
              1. Push the module in one of the possible
              directions.
              2. Change the connection face.
              3. Change the orientation by doing a rotation
              in a certain axis.
      else
        break
      end if
    end for
    Check (child of m)
  end for
end procedure
```

Since this procedure is a sequence of actions where there is no randomness, the same genotype will produce always the same phenotype. The algorithm, as it can be applied to new phenotypic robots that were randomly generated, it can also be applied to robots generated from genetic operations. When the robot can not be recovered, a new genotype is created.

While doing experimentations, 3,39% of unsuitable morphologies were generated. The recovery rate was 71,56%.

2.2 Control System

The control system allows us to control the robot joints at every time step by applying torques. Each module has its own local controller, here based on a Multi-Layer Perceptron (MLP) [7]. It controls the joint that links the module to its parent. Hence, root node has no local controller. Duplicated modules that have been created from the same graph node will have the same controller. Modules can communicate with each other and can receive orders from a root module using embedded communication. Therefore, the global behavior of a robot emerges from the collaboration between all those local behaviors. An example of the distributed control system is illustrated in figure 2.

At the genotype level, all generated controllers have the same number of neurons in the input layer. These neurons have as input the data gotten from sensors and the data gotten from neighborhood communication. However, at the expression of the phenotypic module, the number may vary depending on the integrated sensors. Neurons of output layer stay the same. The number of hidden layers and the number of neurons on each layer is different. On the input layer, one of the neurons is responsible for receiving messages. It receives one message at a time.

Communication neurons are used to exchange messages between neighbors, for instance, a processed information gotten from an eventual sensor. They are also used to send messages to the brain of the robot. The later is a controller (MLP) positioned in the root module. It has the possibility to control all the joints at any time step (not every time step). This depends on the controller's inputs. Input neurons receive information from other modules and general information about the robot, for example the mass and the number of modules. That way, the brain can intervene as soon as there is an unexpected change in the environment (sudden appearance of an obstacle, etc.) or in the morphology of the robot. It's like a reflex. Actually, this functionality may not have an effect on the robot's behavior since the environment is not complex.

2.2.1 Sensors

Each module employs a set of sensors that informs it of the external world as well as its internal state (the state of the module and the state of the joint). Currently, 6 sensors are used:

- joint angle
- joint torque
- module angular and linear velocity
- module orientation (yaw, pitch, roll)
- module force
- contact sensor (in future we will use other sensors)

2.2.1 Effectors

The robots' effectors applies torques into joints. The torque applied to each joint is a sinusoidal function. The first neuron of the output layer is the amplitude and the second is the phase.

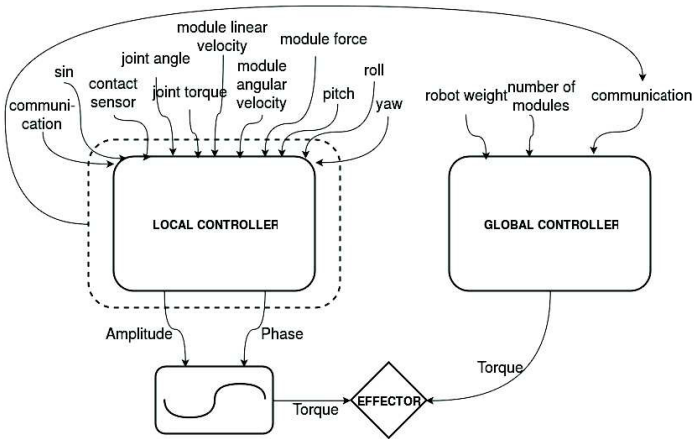

Fig. 3. Local and global controllers with defined inputs and outputs

3 JOINT EVOLUTION

Evolution of virtual robots allows the emergence of the fittest ones that can, in our case, develop locomotive strategies in order to travel a long distance. We use Genetic Algorithms to evolve robots' morphologies along with their controllers. The aim is to optimize the morphology and the neural network parameters in order to obtain the appropriate combination.

Evolution is a cyclic process. At every cycle, from the current generation, new individuals are created (offspring) using genetic operations. These individuals will compose the population of the next generation, and so on. First an initial population of genotypes are created randomly with the following parameters: a graph can contain from 1 to 4 nodes and can have at maximum 3 reflectives and recursive nodes. Recursion can be applied up to 10 times while

reflection can be applied up to 4 times. Genotypes are expressed to produce the corresponding phenotypes. Individuals (phenotypic robots) are then evaluated in a three-dimensional environment.

3.1 Genetic operations

Genetic operations are done at the genotype level of the virtual robot. All genotype parameters can be affected by these operations. Mutation can affect the robot morphology, its controller, or both at the same time. According to a given probability we can either add or remove a graph node and either add or remove a connection (that can be recursive or reflective). We can change the number of applications of a recursive and reflective connections. The node from where generation of the phenotype starts (root node) can be modified. Scaling factor, Reflection, and recursion method can also be changed. For each node, we can change module and joint phenotypic parameters (module size, joint position and rotation axis). Two new robots are created from mating two robots' genotypes (graphtals) selected from the population. We used two methods for mating, crossover and grafting [1]. Robots were selected with the tournament selection method with 7 tournaments.

3.2. Evaluation function

We choose to compute the traveled distance by the robot to evaluate its performance and its ability to perform tasks. Robots were evaluated in the environment for 30s of a simulation time. The initial point is taken after the stabilization of the robot (after 3s).

4 ENVIRONMENT

Robots and their controllers were evaluated one by one in Gazebo simulator. In the next section we will introduce the platform and then we will go into the physical attributes and the realism of the simulation.

4.1. Gazebo

We integrated our system in Gazebo, a popular realistic robotic simulator often used to simulate large robots (humanoids, wheeled-robots, etc.). One of the key component of the simulation platform is the use of simulated sensors that produces a data stream which closely matches data from real-world physical sensors. It provides for the environment and the simulated objects numerous physical attributes that we can set up with relevant realistic values.

The physical description of the environment and all simulated objects in the scene is described in an XML file with the SDF format (Simulation Description Format). Modifying the simulation during run time requires the use of *Plugins*. They are programs written in C++ that can for instance control a robot's joint.

4.2. Physical attributes

To bring more realism to the simulation (visualization and behavior) it is necessary to set up physical attributes with realistic values close to reality.

- Module dimensions: they are ranging from 5 cm to 10 cm. In literature, modules are in that range [12, 17, 18].
- Module mass: the mass is calculated from the volume of the module and its density. Modules have the density of water.
- Module moment of inertia: the moment of inertia determines the difficulty of making an object rotate. It is the most influential parameter on the behavior realism.
- Friction: when applied on module surface, opposing frictional force can avoid robot sliding, and when applied in joints will avoid the unrealistic vibration moves. We set module surface friction to 0,5 and we set joint friction to 0,2.
- Joint damping: depending on joint velocity, damping allows the energy dissipation. This can avoid bouncing movements. After several experimentations we choose 0,02.
- Joint torque: it is the required force that can make a joint rotate and raise all the modules attached to him without breaking up the joint. However we set a torque limitation to 1,75 Nm.
- Joint velocity: the maximum velocity allowed to a joint is 5 rad/s.

Setting this parameter with the maximum possible reality, they are helping to avoid unrealistic robot behavior. We present three cases of unrealistic simulations that we met:

Case A: There is not enough torque applied to joints, the robot remains motionless.

Case B: There is too much torque, the robot bounces randomly and travel a distance that can be too large.

Case C: There is enough torque, but one of the physical attributes lacks of precision, the robot moves by a continuous sliding without rotating the joints. This is caused when a module pushes another module for a long time.

In case B and C, when the fitness computation is based on the travelled distance, the evolution favors the aberrant behavior of robots at the expense of the realistic ones. The more there is restriction and limitation at physical parameters, the more we limit the emergence of particular cases that can skew evolution.

5 EXPERIMENTAL RESULTS

In order to test and evaluate the efficiency of our system we conducted a set of experiments. Each experiment starts with a random generation of 50 robots. Experiments ran for 100 generations. Crossover and mutation rate were respectively 35% and 75%.

We studied the influence of the robot's size (number of modules) on the capacity of locomotions. In the first experiment we had three cases of robots with a maximum number of modules of 5, 10, and 15. The experiment is repeated twenty times for each size.

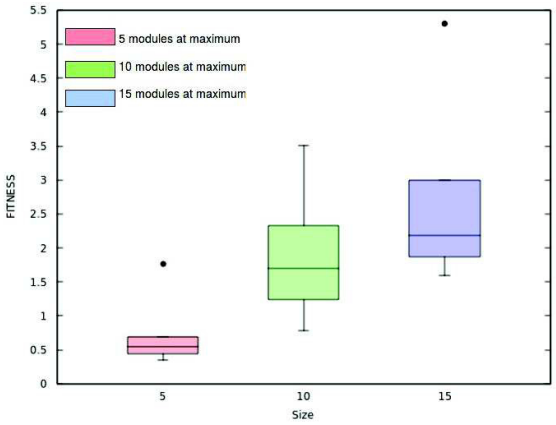

Fig. 4. The influence of the morphology size on the traveled distance. With the defined evolution parameters (maximum torque allowed is 1,75 Nm, up to 10 application of recursion) larger robots perform better. The box plots show the median, interquartile range, min and max values.

Graph in figure 4 shows that, with our system, robots with bigger number of modules traveled more distance. Smaller robots with 5 maximum modules traveled in average 0,55 meter while larger ones with 10 and 15 maximum modules could travel respectively 1,7 meter and 2,18 meters in 30 seconds. We had noticed that the number of modules of the fittest robot on each run is equal to the maximum allowed modules. Large evolved robots in the two last cases were snake-like, some of theme developed jumping behaviors. So with their long length (can go to 1 m), big jumps allowed them to cross long distances. Therefore smaller robots can not survive even if they perform well. Snake-like robots were favored because we allowed recursion up to 10 times and torque up to 1,75 Nm. This torque value was high enough to make those large robots move. But theoretically larger robots are not necessary the more suitable for long distance locomotion. Smaller robots with light weight can go fast and therefore travel more distance. But this is not the case with this experiment: smaller robots traveled less distance comparing them larger ones even though the required toque was sufficient. They were either heavy or their morphologies were simple in a way that they can not allow effective moves.

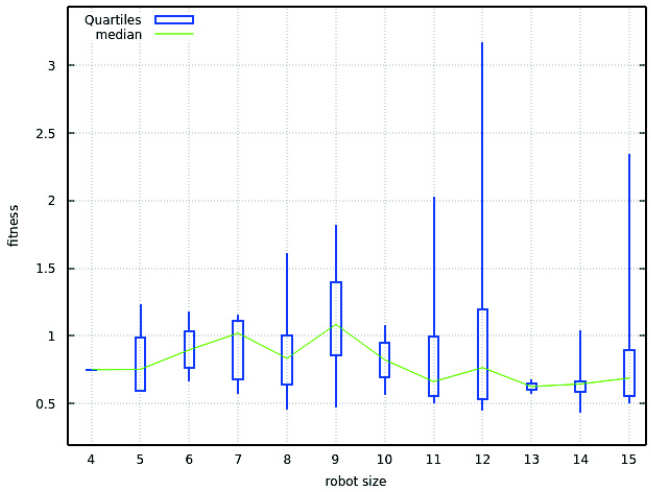

Fig. 5. The influence of the morphology size on the traveled distance. Evolution parameters are (maximum torque allowed is 0,17 Nm, up to 2 application of recursion). The box plots show the median, interquartile range, min and max values.

In this next experiment we wanted to give a chance to smaller robots to evolve in a way that larger robots can too (by not letting snake like emerge). We reduced the maximum recursion application to 2. This way larger robots' morphologies will be more varied (not necessary snake-like). The effort limit in joints was 0,17 Nm. The maximum number of allowed modules in a robot was 15. We repeated the experiment 80 times where in each run the number of modules of the fittest robot may be from 1 to 15. In the table below, we show the number of evolved robots for each robot size.

Table 1. Size of evolved robots

Robot's size	**4**	**5**	**6**	**7**	**8**	**9**
Evolved robots	1	4	7	5	13	8
Robot's size	**10**	**11**	**12**	**13**	**14**	**15**
Evolved robots	2	8	15	2	5	10

Graph in figure 5 shows that robots with 9 modules are in general better at long distance traveling. Robots with more then 10 modules were in average less efficient. This may be caused by the torque limitation. Motors were not powerful enough to make the joints move. However, if the morphology of these larger robots allow effective moves and their module's weights are light they can travel very long distances.

Experimental results show that indeed robots' size has an affect on the traveled distance. The more there are modules the more varied interesting morphologies can be

generated, and therefore a possibility to have effective behavior that allow long distance locomotion. In fact, we believe that larger robots are more appropriate for long distance traveling, we only need to provide the necessary torque for each joint.

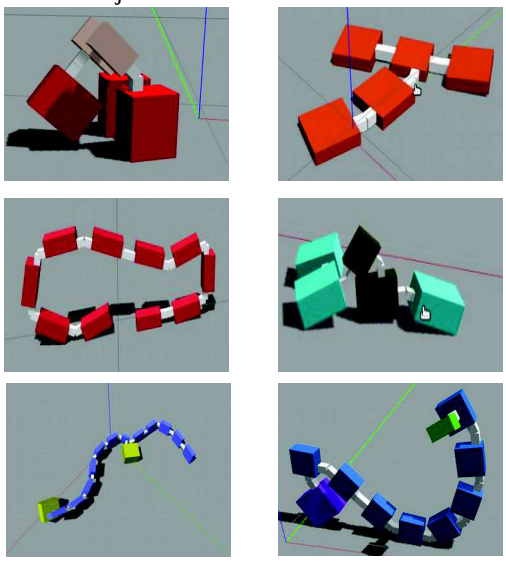

Fig. 6. Evolved robots examples. Each one developed a strategy to do locomotion.

6 CONCLUSION

In this paper we have introduced our system that can generate realistic heterogeneous modular robots in a 3D realistic environment. Integrating our system in Gazebo and having access to set up numerous physical parameters lead to obtain realistic simulations. We expect the virtual robots generated with our system and trained in a realistic robotic simulator to better bridge the gap to reality. Our system generated interesting robots that learned to find locomotion strategies in order to travel longest distances. In the future we will improve our model in order to generate robots that can do complex tasks in complex environments where there are obstacles and uncertainties. For that, we will try to add other module shapes, we will implement a more efficient controller, and we will add other sensing capabilities.

REFERENCES

[1] Sims K (1994), Evolving virtual creatures. Proceedings of SIGGRAPH, New York, USA, 1994, pp.15-22

[2] Sims K (1994), Evolving 3d morphology and behavior by competition. Artificial Life IV Proceedings, MIT Press, Cambridge, 1994, pp.28-39

[3] Bongard JC, Bernatskiy A, Livingston K, Livingston N, Long J, Smith M (2015), Evolving robot morphology facilitates the evolution of neural modularity and evolvability. GECCO Proceedings , Madrid, 2015, pp.129-136

[4] Cappelle CK, Bernatskiy A, Livingston K, Livingston N, Bongard JC (2016), Morphological modularity can enable the evolution of robot behavior to scale linearly with the number of environmental features. Frontiers in Robotics and AI, 3(59)

[5] Yim M, Shen WM, Salemi B, Rus D, Moll M, Lipson H, Klavins E, Chirikjian G (2007), Modular self-reconfigurable robot systems. IEEE Robotics & Automation Magazine, 14(1), pp.43-52

[6] Gilpin K, Rus D (2010), Modular robot systems. IEEE Robotics & Automation Magazine, 17(3), pp.38–55

[7] Hagan MT, Demuth HB, Beale M (1996), Neural network design. Boston, PWS Publishing Company

[8] Aiben AE, Smith J (2015), From evolutionary computation to the evolution of things. Nature, 521, pp.476-482

[9] Lipson H, Pollack JB (2000), Automatic design and manufacture of robotic lifeforms. Nature, 406, pp.974–978

[10] Krcah P (2007), Evolving virtual creatures revisited. GECCO '07 Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, New York, USA, pp. 341

[11] Koenig N, Howard A (2004), Design and use paradigms for Gazebo, an open-source multi-robot simulator. In IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, Sep 2004, pp. 2149-2154

[12] Auerbach J, Aydin D, Maesani Andrea, Kornatowski P, Cieslewski T, Heitz G, Fernando P, Loshchilov Ilya, Daler L, Floreano D (2014), RoboGen: robot generation through artificial evolution. Artificial Life 14: International Conference on the Synthesis and Simulation of Living Systems, New York, NY, USA, July 30-August 2, 2014

[13] Lassabe N, Luga H, Duthen H (2007), A new step for artificial creatures. In Proceedings of 1st IEEE Conference on Artifcial Life, pp.243-249

[14] Chaumont N, Egli R, Adami C (2007), Evolution of virtual catapults. Artificial Life, 13(2), pp.139–157

[15] Hornbey GS, Pollack JB (2001), Evolving L-system to generate virtual creatures. Computers and Graphics, 25(6), pp.1041-1048

[16] Nol S, Floreano D (2000), Evolutionary robotics. MIT Press, Boston, MA

[17] Davey J, Kwok N, Yim M (2012), Emulating self-reconfigurable robots - design of the SMORES system, in Intelligent Robots and Systems, 2012, pp.4464-4469

[18] Cussat-Blanc S, Pollack J (2012), A cell-based developmental model to generate robot morphologies. In Proceedings of the 2012 Genetic and evolutionary computation conference (GECCO), pp.2549-2556.