Simulating the Evolution of Soft and Rigid-Body Robots

Sam Kriegman
MEC Lab§
University of Vermont
Burlington, VT, USA
sam.kriegman@uvm.edu

Collin Cappelle*
MEC Lab
University of Vermont
Burlington, VT, USA
collin.cappelle@uvm.edu

Francesco Corucci
The BioRobotics Institute
Scuola Superiore Sant'Anna
Pisa, Italy

Anton Bernatskiy
MEC Lab
University of Vermont
Burlington, VT, USA

Nick Cheney
Creative Machines Lab
Cornell University
Ithaca, NY, USA

Josh C. Bongard
MEC Lab
University of Vermont
Burlington, VT, USA

ABSTRACT

In evolutionary robotics, evolutionary methods are used to optimize robots to different tasks. Because using physical robots is costly in terms of both time and money, simulated robots are generally used instead. Most physics engines are written in C++ which can be a barrier for new programmers. In this paper we present two Python wrappers, Pyrosim and Evosoro, around two well used simulators, Open Dynamics Engine (ODE) and Voxelyze/VoxCAD, which respectively handle rigid and soft bodied simulation. Python is an easier language to understand so more time can be spent on developing the actual experiment instead of programming the simulator.

KEYWORDS

Physical simulation; Evolutionary robotics.

ACM Reference format:
Sam Kriegman, Collin Cappelle, Francesco Corucci, Anton Bernatskiy, Nick Cheney, and Josh C. Bongard. 2017. Simulating the Evolution of Soft and Rigid-Body Robots. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082051

1 INTRODUCTION

Physical simulation provides a relatively inexpensive, surrogate medium for the optimization of robot controllers and hardware. Simulation is particularly important in evolutionary robotics which generally makes fewer assumptions concerning the structure of a robot's 'brain' (its controller) and body plan. Unfortunately, however, physics simulators are not well maintained, and can be difficult to learn and extend by modifying existing code which raises the barrier of entry, particularly for students. For this reason we introduce here two high-level Python wrappers around two different simulators: Open Dynamics Engine (ODE) and Voxelyze. Our goal in this paper is to convey from experience what kinds of things are difficult/easy to instantiate in physics engines in general and how our user-friendly modules at least partially alleviate this, and how they may be extended in the future through open-source collaborations.

*The first two authors contributed equally to this work.
§The Morphology, Evolution & Cognition Laboratory (**www.meclab.org**).

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3082051

2 RIGID BODY ROBOTS

Rigid body dynamics engines are generally what is most thought of when one thinks of a simulator. They are 3D engines where every body in the simulation is 'rigid', meaning the body cannot bend, break or change as a result of contact with other bodies in the system. These simulators usually provide methods for creating differently shaped bodies (i.e. cylinders, spheres, boxes, etc.), joints between the bodies, and ways to actuate the motors. This makes them ideal for most robot simulation because most real world robots are still made of hard materials like metal.

ODE is an open source rigid body simulator written in C++ used prevalently in the robotics community because of its speed and stability [13]. There are many other simulators who use ODE's collision detection and response as the basis for their engines. However, most of these extensions of ODE continue to use C++ which makes them difficult to pick up quickly for new users or people who are unfamiliar with simulators. There is a lack of easy-to-use simulators for 3D rigid body dynamics.

2.1 Pyrosim

Pyrosim is a Python package used to send robots to an ODE simulation and get back the resulting sensor data of that robot [1]. The goal of Pyrosim is to limit how much the user has to deal with ODE and C++ code. By focusing on ease of use, Pyrosim can greatly limit the amount of developmental time spent building different robots while still maintaining evaluation speed because the underlying simulation runs on ODE. This is especially useful for users who are new to simulators or to coding in general because Python is an easier language to learn than C++. For example, figure 1 shows the necessary code to make a simple two cylinder robot connected by a hinge joint and moves based on response to touch sensors in the cylinders. The corresponding simulated robot is shown in figure 2.

After evaluation, the user can collect the results of the simulation in the form of a numpy matrix which contains the value of each sensor in the robot at every time step of simulation meaning a wide

```
from pyrosim import PYROSIM

ARM_LENGTH = 0.5
ARM_RADIUS = ARM_LENGTH / 10.0

#create simulation instance
sim = PYROSIM(playPaused = False , evalTime = 1000)
#Make cylinders in ODE
sim.Send_Cylinder(objectID = 0 , x=0, y=0, z=ARM_LENGTH
    /2.0 + ARM_RADIUS, r1=0, r2=0, r3=1, length=
    ARM_LENGTH, radius=ARM_RADIUS)
sim.Send_Cylinder(objectID = 1 , x=0, y=ARM_LENGTH/2.0, z
    =ARM_LENGTH + ARM_RADIUS, r1=0, r2=1, r3=0, length=
    ARM_LENGTH, radius=ARM_RADIUS)
#Joint between the cyliders
sim.Send_Joint(jointID = 0, firstObjectID=0,
    secondObjectID=1, x=0, y=0, z=ARM_LENGTH +
    ARM_RADIUS, n1=1, n2=0, n3=0, lo=-3.14159/4.0, hi
    =+3.14159/4.0)
#touch sensors on each cylinder
sim.Send_Touch_Sensor(sensorID = 0 , objectID = 0)
sim.Send_Touch_Sensor(sensorID = 1 , objectID = 1)
#sensor neurons for each sensor
sim.Send_Sensor_Neuron(neuronID=0, sensorID=0 )
sim.Send_Sensor_Neuron(neuronID=1, sensorID=1 )
#motor neuron which moves the hinge joint
sim.Send_Motor_Neuron(neuronID=2 , jointID=0 )
#Synapses directly connect sensor neurons to motor neuron
sim.Send_Synapse(sourceNeuronID = 0 , targetNeuronID = 2
    , weight = 1.0 )
sim.Send_Synapse(sourceNeuronID = 1 , targetNeuronID = 2
    , weight = -1.0 )

#start the simulation and get back touch sensor data
sim.Start()
sim.Wait_To_Finish()
sensor_data = sim.Collect_Sensor_Data()
```

Figure 1: Pyrosim demo code

range of evaluation functions can be performed. Built-in sensors include touch sensors, position sensors, light sensors, and many more. By using a class structure, e.g. objects to represent robots in Python, a user can easily send multiple different robots into the same environment as seen in figure 3. Here different individuals in the population were competing in the same environment to see which one moved the furthest [1]. While Pyrosim can easily handle small swarms of robots it should not be used for very large (40+) swarms of robots due to limits to the number of bodies, synapses, and neurons which can be sent to ODE and ODE's own constraints on the number of objects in simulation.

Pyrosim contains most of what is necessary to create arbitrary robot morphologies, within the limits of ODE. Controllers in Pyrosim are more limited. They must be neural networks made up of sensor, motor, hidden and bias neurons whose activation function is detailed by Eq. 1.

$$y_i^{(t)} = \tanh\left(y_i^{(t-1)} + \tau \sum_{j \in J} w_{ji} y_j^{(t-1)}\right) \tag{1}$$

Where

- $y_i^{(t)}$ is the value of neuron i at time step t
- w_{ji} is the weight of the synapse from neuron j to i
- τ is the learning rate in the network

[1] **https://youtu.be/ecSI9CqZI1E**

This limits the type of network can be used without editing the C++ code.

Extending Pyrosim is relatively straightforward assuming the user knows C++ and ODE. Pyrosim exchanges strings with the C++ code through standard input and output. On the Python side, the subprocess module is used to communicate with the simulator. The C++ side reads in commands using `std::cin`. These commands are then compared with predefined strings to find the corresponding C++ process that needs to be run to change the simulation. Thus, extending Pyrosim to implement a new feature amounts to creating a new output command on the Python side, a corresponding input catch on the C++ side, and whatever function implements the new feature in C++.

Pyrosim is a rigid-body simulator with a focus on being easy to develop new robots. It is particularly useful for users who are new to simulators or programming. Many more features are available in Pyrosim than what is presented here which help a user to easily make complex robots in order to evolve interesting behaviors. Pyrosim also provides a framework for relatively easy expansion provided the user has some experience with ODE and C++. Further development for Pyrosim will aim at creating a 'starter kit' for evolutionary robotics, which will provide a user all they need to start their own experiments simulating evolving robots.

3 SOFT ROBOTS

Soft robots provide numerous advantages over their rigid counterparts including many more (theoretically infinite) degrees of freedom and compliant structures which facilitate unloading expensive computation from the controller to the body, i.e. *morphological computation* [9]. Beyond morphological computation, this structural flexibility enables soft robots to continuously modify their form while they behave which in turn allows soft robots to perform tasks that rigid bodies could not— such as conforming to uneven surfaces, dampening or amplifying vibrations, efficiently distributing stress, morphing to meet different tasks, and squeezing through small apertures (figure 4). However there are several challenges to the field of soft robotics, including limited simulation and design automation tools [12].

The dynamics of soft materials are difficult and computationally expensive to simulate given their many degrees of freedom and nonlinear geometric deformations. Standard nonlinear finite element solvers fall short of quantitatively simulating all but relatively small deformations. To address this, Hiller & Lipson [10] developed a computationally efficient approach based on nonlinear relaxation, and introduced the open-source *Voxelyze*, a voxel-based soft matter physics engine and the corresponding (and also open-source) graphical user interface, *VoxCAD*. Voxelyze simulates elastic voxels based on an internal lattice of discrete points (with mass and rotational inertia) connected by spring-like beam elements (with translational and rotational stiffness) generating realistic, and possibly very large deformations under applied forces. Henceforth, we will refer to both the GUI and the underlying physics engine as simply *VoxCad*.

Robots in VoxCad are constructed from voxels — analogous to the cubic building blocks used in Minecraft — with specified material properties such as stiffness and volume. Robots may be volumetrically actuated with an arbitrary phase offset (at the voxel level) from

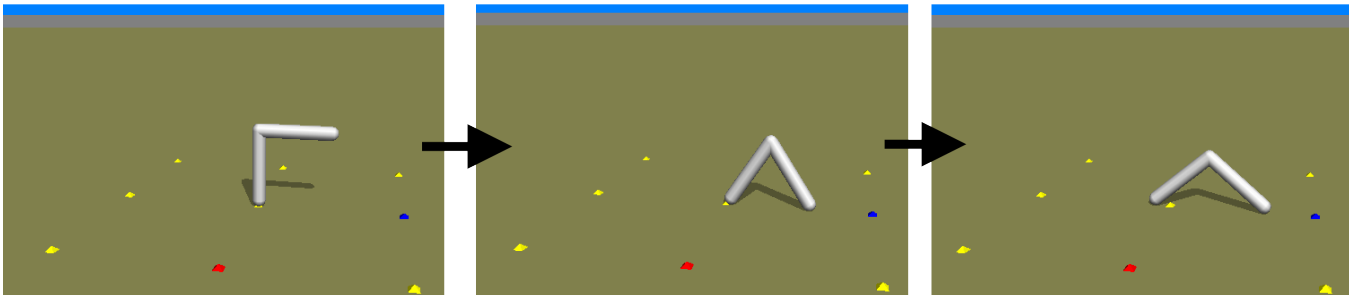

Figure 2: Pyrosim demo in simulation. The robot consists of two cylinders connected by a hinge joint controlled by a simple neural network. The behavior was not evolved so the robot simply actuates and falls over.

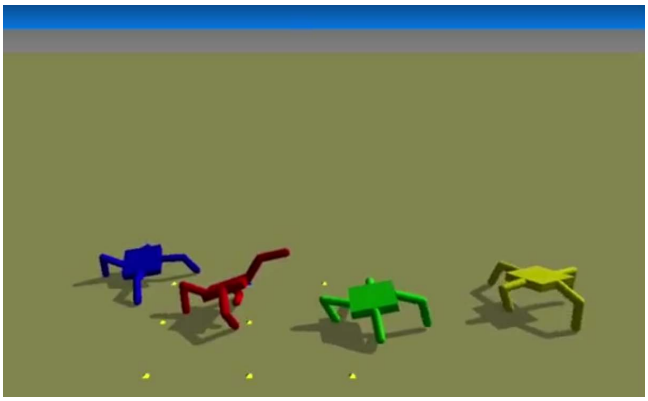

Figure 3: Multiple evolved quadrupeds in the same Pyrosim simulation

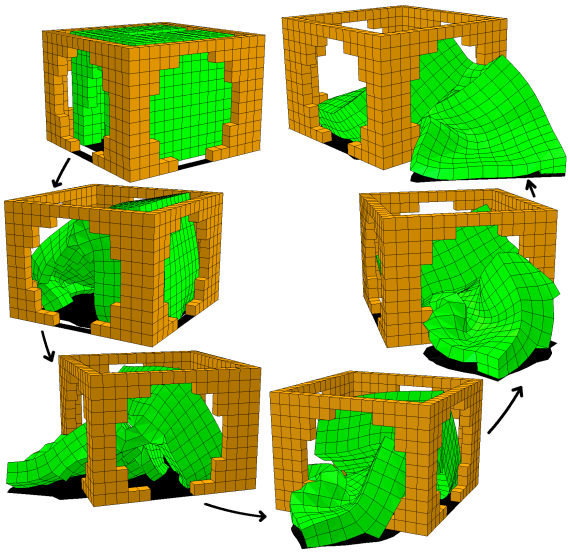

Figure 4: A soft robot, simulated in VoxCad, escaping from a cage (reprinted from Cheney et al. [3]).

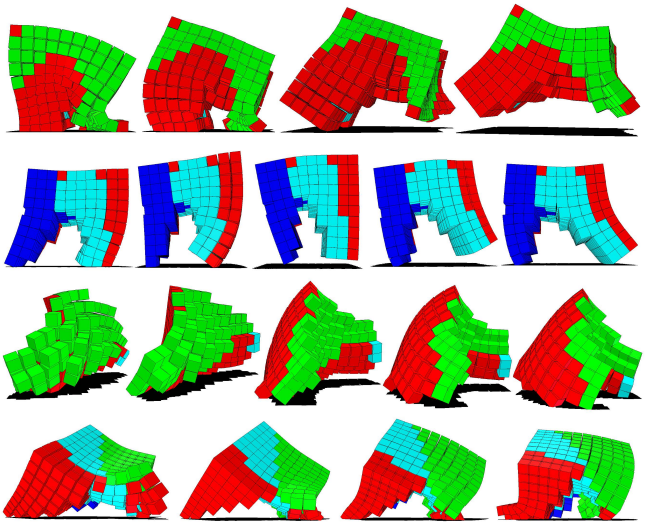

Figure 5: Four different soft robots moving with various gaits from left to right (reprinted from Cheney et al. [5]).

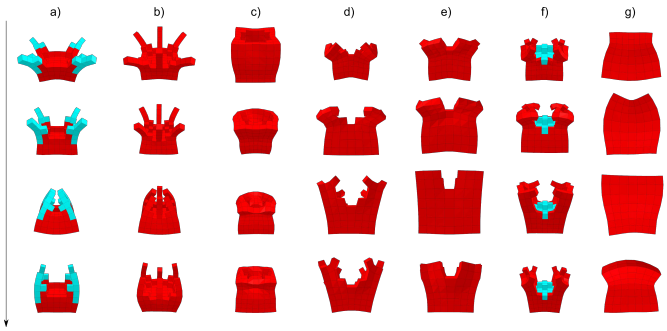

Figure 6: Seven different soft-bodied creatures evolved underwater, swimming from top to bottom (reprinted from Corucci et al. [7]).

an oscillating global signal. In previous work[2], robots have been designed to locomote either with patches of voxels composed of 'muscle' material that actuates in counter-phase with complementary patches of 'muscle' [5], or via propagating waves of actuation [4, 7]. Robots can also stretch and reach towards an object by changing voxel volumes [8, 11] which could in principle induce locomotion via peristalsis.

3.1 Evosoro

While it is relatively easy to read a fixed body plan of a robot into VoxCad, it can be tedious to edit these blueprints and there are a several algorithmic caveats in their automated optimization. VoxCad, like ODE, is a general-purpose physics engine and thus does not include any optimization tools which are inherently job-specific. Moreover, controlling soft robots is non-trivial, as motion at one location can propagate in unanticipated ways to other parts of the body [2]. For these reasons, in a similar fashion as Pyrosim, we created a Python wrapper around Voxelyze. *Evosoro*, as the name implies, is a module[3] facilitating the evolution of soft robots (for a brief overview on the evolution of soft robots see [6]).

Evosoro supplies a powerful evolutionary algorithm, developed by Cheney et al. [5] which is highly modularized and adjustable. Various examples using the default subprocesses are provided so the user can immediately begin, out-of-the-box, to optimize running, swimming, and growing soft-bodied creatures. The wrapper primarily maintains an indirect encoding via Compositional Pattern Producing Networks (CPPNs, [14]) which, by default, dictate the placement of voxels as building blocks and their material properties. Multi-network-CPPN genotypes are written to individual robot files which are sent in batch to Voxelyze for headless simulation. In addition to the wrapper, we have empowered the physics engine with new features like a fluid model for evolving swimming creatures [7] and developmental models for adapting morphology during the evaluation, both in open-loop [11] and based on proprioceptive/exteroceptive stimuli [8]. For now, we have adopted a plug-and-play model in which multiple versions of Voxelyze are included and may be referenced by the single wrapper in different scripts. The plug-and-play model is not ideal, but it keeps the repository up to date while avoiding complications related to merging code in development across many different research projects; and the wrapper unifies these different version under a single system.

Entire experiments can be written in short scripts which are easily adjustable and may be stopped and restarted at any generation, thanks to a built-in checkpointing mechanism. Open loop controllers dictating phase offset can be evolved without touching the simulator (completely in Python), but closed loop controllers or any mid-simulation changes to the robot's brain or body plan need to be manually added (in both C++ and Python) at this point. However we provide a few examples of this. Crucially, there is no mechanism (yet) for automatically sending neural networks from Python to Voxelyze. Future work will involve incorporating a mechanism for easily adjusting closed-loop controllers and further working to limit the frequency with which a user would have to touch VoxCad's C++ code.

[2]**https://goo.gl/YHBG0U** links to a YouTube playlist featuring a variety of examples of soft robots evolved with VoxCad.

[3]**https://github.com/skriegman/evosoro** contains the Evosoro repository.

4 CONCLUSION

Physical simulation is critical in evolutionary robotics but there can be a high bar of entry for experts and students alike. To address this, we introduced two high-level, open-source Python libraries — Pyrosim and Evosoro — which facilitate the automatic design and optimization of both soft and rigid-bodied robots. They retain most of the speed because the underlying engine remains the same meaning the bottleneck in total evaluation time is still the actual simulation of the robot. The real trade-off in these simulators is usability verses control and features. These simulators impose some restrictions and do not retain all features in order to make the developmental start-up cost low and simulators more accessible.

5 ACKNOWLEDGEMENTS

NSF awards PECASE-0953837 and INSPIRE-1344227, as well as the Army Research Office contract W911NF-16-1-0304, provided financial support to develop both simulators described herein. F. Corucci is supported by grant agreement #604102 (Human Brain Project) funded by the European Union Seventh Framework Programme (FP7/2007-2013). N. Cheney is supported by NASA Space Technology Research Fellowship #NNX13AL37H. We also acknowledge computation provided by the Vermont Advanced Computing Core.

REFERENCES

[1] Josh Bongard. 2017. Pyrosim. (2017). https://jbongard.github.io/pyrosim/
[2] Josh C Bongard. 2013. Evolutionary robotics. *Commun. ACM* 56, 8 (2013), 74–83.
[3] Nick Cheney, Josh Bongard, and Hod Lipson. 2015. Evolving soft robots in tight spaces. In Proceedings of the 2015 annual conference on Genetic and Evolutionary Computation. ACM, 935–942.
[4] Nick Cheney, Jeff Clune, and Hod Lipson. 2014. Evolved electrophysiological soft robots. In ALIFE, Vol. 14. 222–229.
[5] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. 2013. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, 167–174.
[6] Francesco Corucci. 2017. Evolutionary Developmental Soft Robotics: Towards Adaptive and Intelligent Soft Machines Following Nature's Approach to Design. In Soft Robotics: Trends, Applications and Challenges. Springer, 111–116.
[7] Francesco Corucci, Nick Cheney, Hod Lipson, Cecilia Laschi, and Josh Bongard. 2016. Evolving swimming soft-bodied creatures. In ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, Late Breaking Proceedings. 6.
[8] Francesco Corucci, Nick Cheney, Hod Lipson, Cecilia Laschi, and Josh Bongard. 2016. Material properties affect evolution's ability to exploit morphological computation in growing soft-bodied creatures. In ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems. MIT press, 234–241.
[9] Helmut Hauser, Rudolf Marcel Füchslin, and Rolf Pfeifer. 2014. Opinions and Outlooks on Morphological Computation. Zürich. 244 pages.
[10] Jonathan Hiller and Hod Lipson. 2014. Dynamic simulation of soft multimaterial 3d-printed objects. Soft Robotics 1, 1 (2014), 88–101.
[11] Sam Kriegman, Francesco Corucci, Nick Cheney, and Josh Bongard. 2017. A Minimal Developmental Model Can Increase Evolvability in Soft Robots. In Proceedings of the 2017 annual conference on Genetic and Evolutionary Computation. ACM.
[12] Hod Lipson. 2014. Challenges and opportunities for design, simulation, and fabrication of soft robots. Soft Robotics 1, 1 (2014), 21–27.
[13] Russell Smith. 2006. Open Dynamics Engine. (2006). http://www.ode.org
[14] Kenneth O. Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines 8, 2 (2007), 131–162.