

# **Modern Robotics: Evolutionary Robotics**

COSC 4560 / COSC 5560

Professor Cheney  
4/27/18

# **Crossing the Reality Gap**

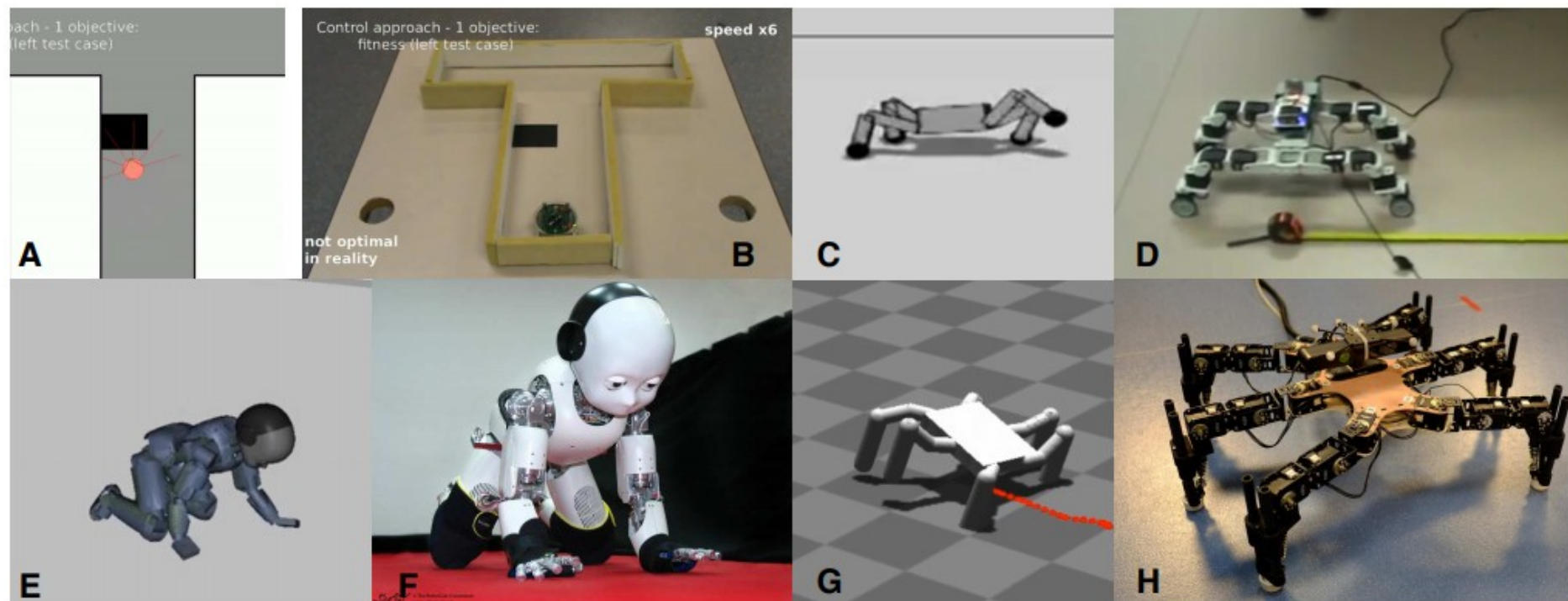


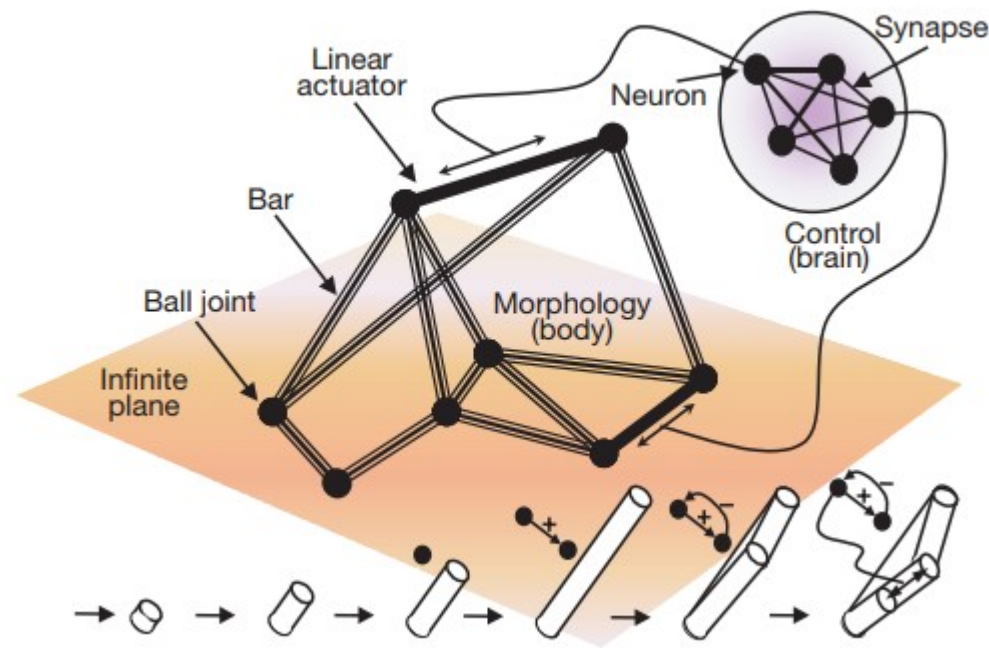
Figure 1: Examples of simulated and real robots we have used in our research. A. Fastsim (fast 2-D kinematic simulator) [16]. B. E-puck robot in the same maze as in A [16]. C. Quadruped robot simulated in RobDyn (Bullet) [14–16, 22]. D. Real quadruped robot based on Dynamixel actuators (AX-12) [14–16, 22]. E. iCub robot crawling simulated in our new Dart-based simulator [23]. F. Real iCub robot crawling. G. Hexapod robot simulated in RobDyn (ODE) [4–6, 13]. H. Real hexapod robot [3–6, 13].

# **Automatic design and manufacture of robotic lifeforms**

**Hod Lipson & Jordan B. Pollack**

*Computer Science Department, Volen Center for Complex Systems,  
Brandeis University, Waltham, Massachusetts 02454, USA*





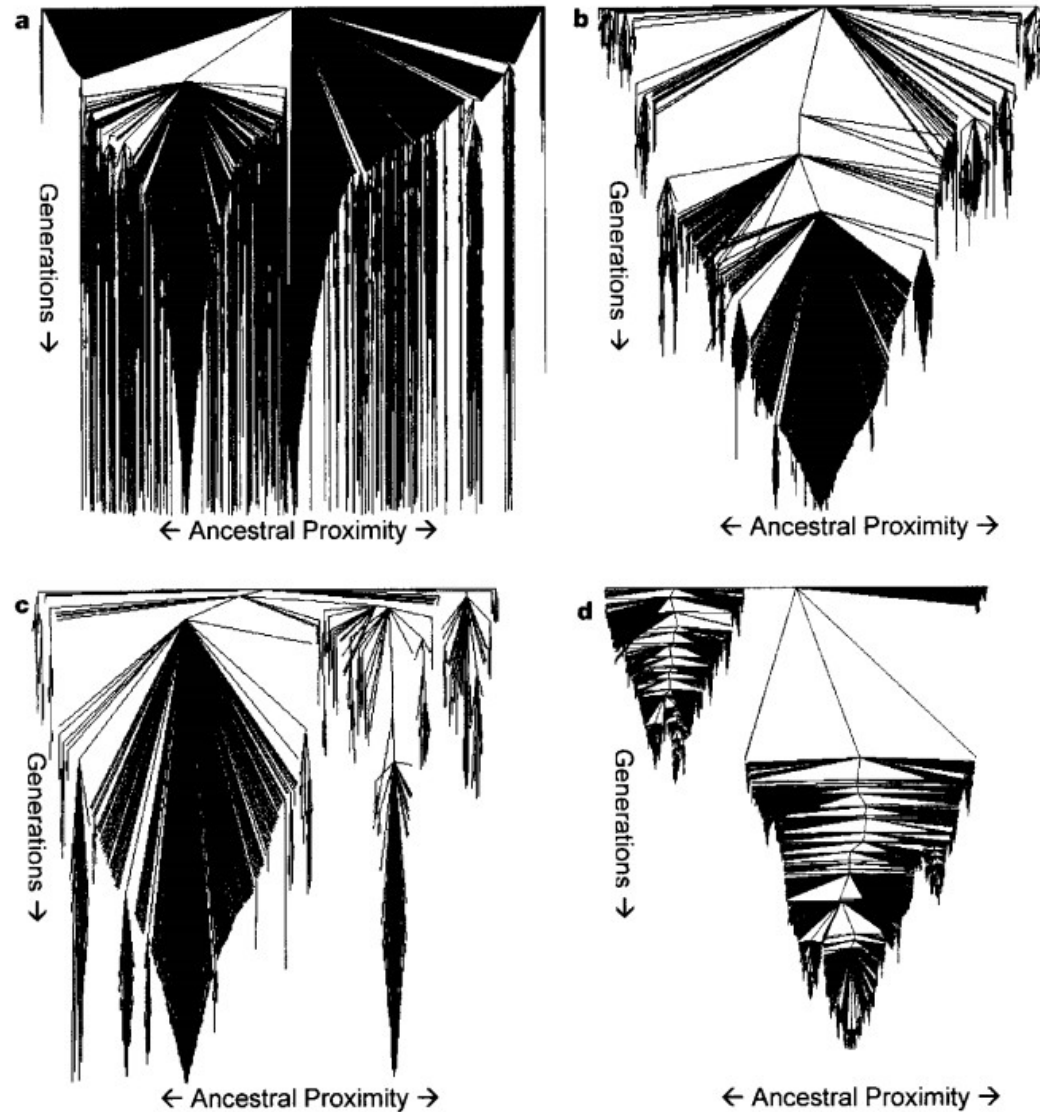
**Figure 1** Schematic illustration of an evolvable robot. Bars connect to each other to form arbitrary trusses; by changing the number of bars and the way they connect, the structural behaviour of the truss is modified—some substructures may become rigid, while others may become articulated. Neurons connect to each other via synapses to form arbitrary recurrent neural networks. By changing the synapse weights and the activation threshold of the neuron, the behaviour of the neuron is modified. By changing the number of neurons and their connectivity, the behaviour of the network is modified. Also, we allow neurons to connect to bars: in the same way that a real neuron governs the contraction of muscle tissue, the artificial neuron signal will control the length of the bar by means of a linear actuator. All these changes can be brought about by mutational operators. A sequence of operators will construct a robot and its controller from scratch by adding, modifying and removing building blocks. The sequence at the bottom of the image illustrates an arbitrary progression of operators that create a small bar, elongate it and split it. Simultaneously, other operators create a neuron, add another neuron, connect them in a loop, and eventually connect one of the neurons to one of the bars. The bar is now an actuator. Because no sensors were used, these robots can only generate patterns and actions, but cannot directly react to their environment.

## Evolution process

Experiments were performed using version 1.2 of GOLEM (Genetically Organized Lifelike Electro Mechanics), which is available at <http://www.demo.cs.brandeis.edu/golem>. We performed a simulated evolutionary process: the fitness function was defined as the net euclidean distance that the centre-of-mass of an individual moves over a fixed number (12) of cycles of its neural control. We started with a population of 200 null (empty) individuals. Each experiment used a different random seed. Individuals were then selected, mutated, and replaced into the population in steady state as follows: the selection functions we tried were random, fitness-proportionate or rank-proportionate. The mutation operators used to generate an offspring were independently applied with the following probabilities: a small mutation in length of bar or neuron synaptic weight (0.1), the removal or addition of a small dangling bar or unconnected neuron (0.01), split vertex into two and add a small bar, or split bar into two and add vertex (0.03), attach or detach neuron to bar (0.03). At least one mutation was applied. The mutations took place on the symbolic representation of the phenotype. After mutation, a new fitness was assigned to the individual by means of a simulation of the mechanics and the control (see details below). The offspring was inserted into the population by replacing an existing individual. The replacement functions we tried chose individuals to replace either randomly, in inverse-proportion to their fitness, or using similarity-proportionate criteria (deterministic crowding<sup>22</sup>). Various permutations of selection-replacement methods are possible; the results we report here were obtained using fitness-proportionate selection and random replacement. However, using rank selection instead of fitness-proportionate selection, or using random selection with fitness-proportionate replacement yielded equivalent results. The process continued for 300 to 600 generations (approximately  $10^5$  evaluations overall). The process was performed both serially and in parallel (on a 16-processor computer). On parallel computers we noticed an inherent bias towards simplicity: simpler machines could complete their evaluation sooner and consequently reproduce more quickly than complex machines (this could be avoided with a generational implementation).

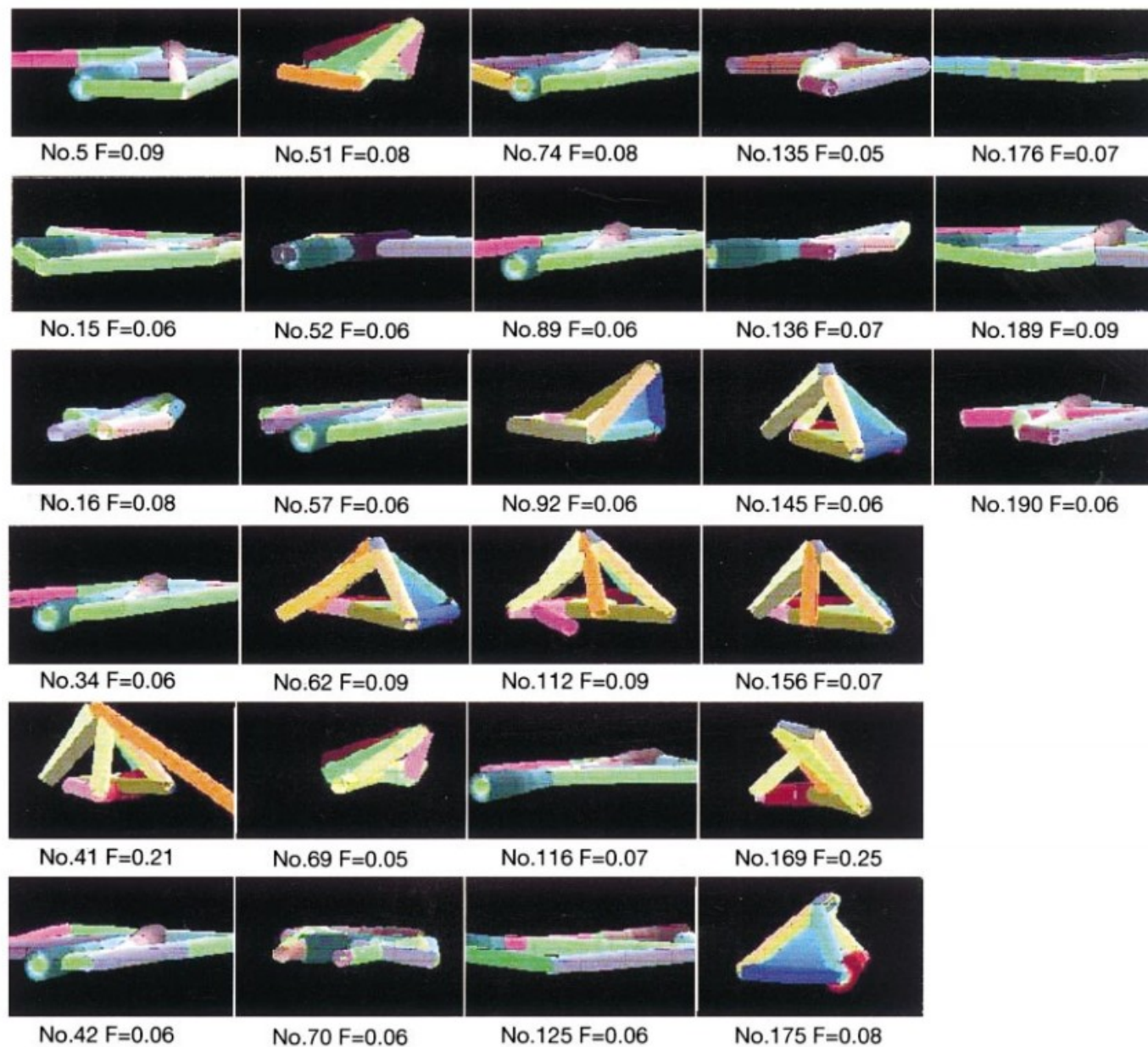
Our evolutionary simulation was based on evolutionary strategies<sup>23</sup> and evolutionary programming<sup>24</sup>, because it directly manipulated continuous valued representations and used only elementary operators of mutation. Alternatively, we could have used genetic algorithms<sup>25</sup> and genetic programming<sup>26</sup> that introduce crossover operators that are sensitive to the structure of the machines, which might change the rate of evolution and lead to replicated structures. We did not form a morphological grammar from which the body is developed<sup>27</sup>, but evolved directly on the symbolic representation of the phenotype. And, instead of separating body (morphology) and brain (control) into separate populations, or providing for a 'neonatal' stage that might allow us to select for brains that are able to learn to control their bodies, we simply applied selection to bodies and brains as integrated units. This simplified experimental set-up followed our focus on completing the simulation and reality loop, but we anticipate that the many techniques that have been developed in evolutionary and co-evolutionary learning<sup>28–30</sup> will enrich our results.





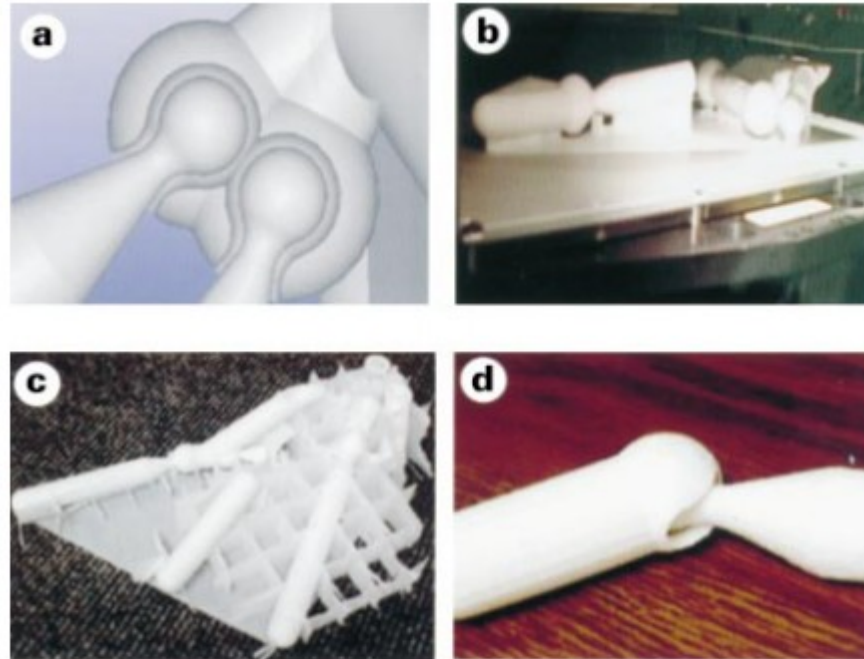
**Figure 2** Phylogenetic trees of several different evolutionary runs. Each node in the tree represents an individual and links represent parent–child relationships. The vertical axis represents generations, and the horizontal axis represents ancestral proximity in terms of the hops along the tree necessary to get from one individual to another. All trees originate at a common root denoting an empty robot with zero bars and actuators. Trees exhibit

various degrees of divergence and speciation: **a**, extreme divergence, resulting from niching methods<sup>22</sup>; **b**, extreme convergence, resulting from fitness-proportionate selection; **c**, intermediate level of divergence, typical of earlier stages of fitness-proportionate selection; and **d**, massive extinction under fitness-proportionate selection. The trees are thinned, and depict several hundred generations each.



**Figure 3** A generation of robots. An arbitrarily sampled instance of an entire generation, thinned down to show only significantly different individuals. The caption under each image provides an arbitrary index number (used for reference) and the fitness of that

individual. Two subpopulations of robots are observable, each with its own variations: one flat on the ground, and the other containing some elevated structure.

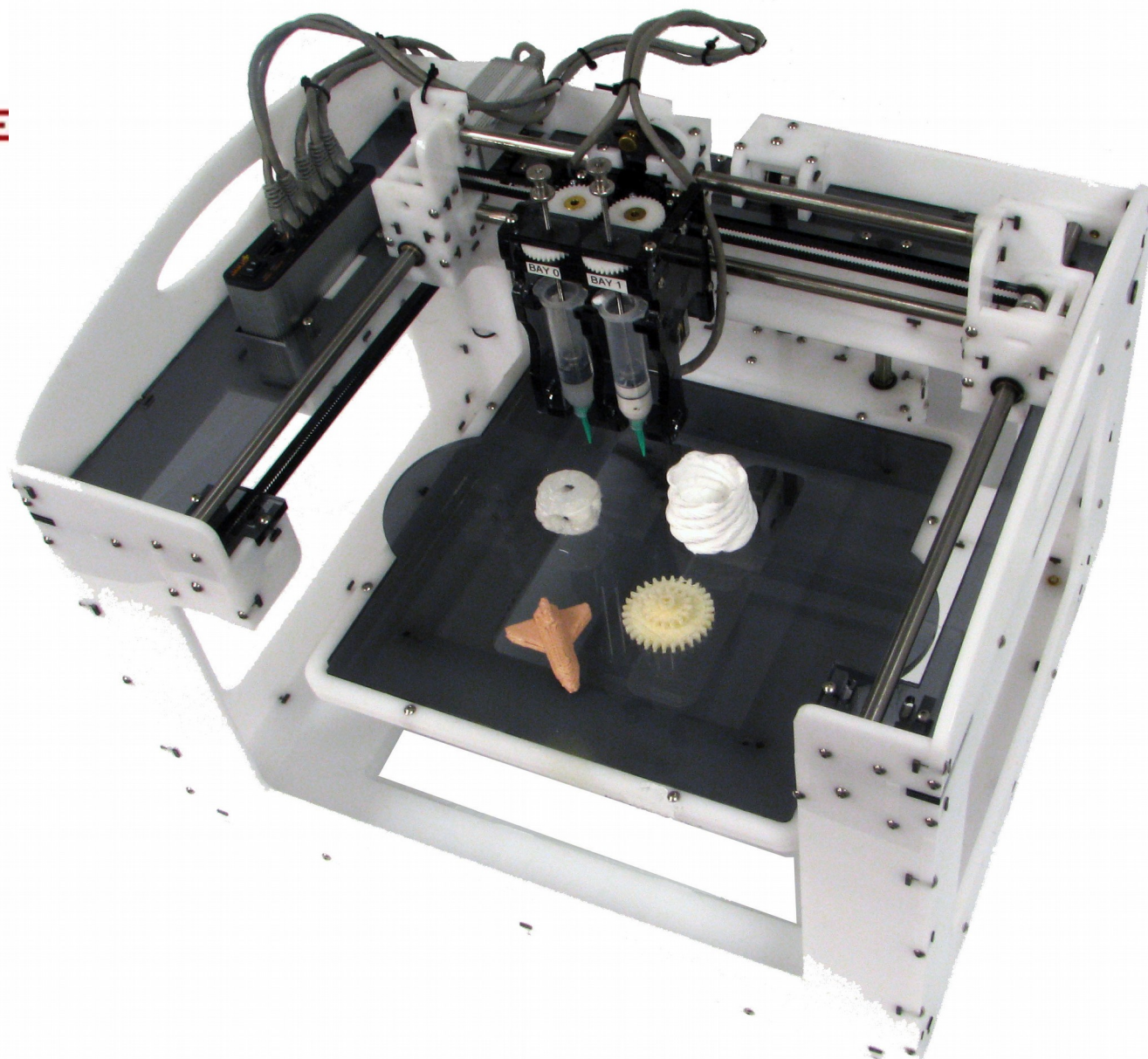


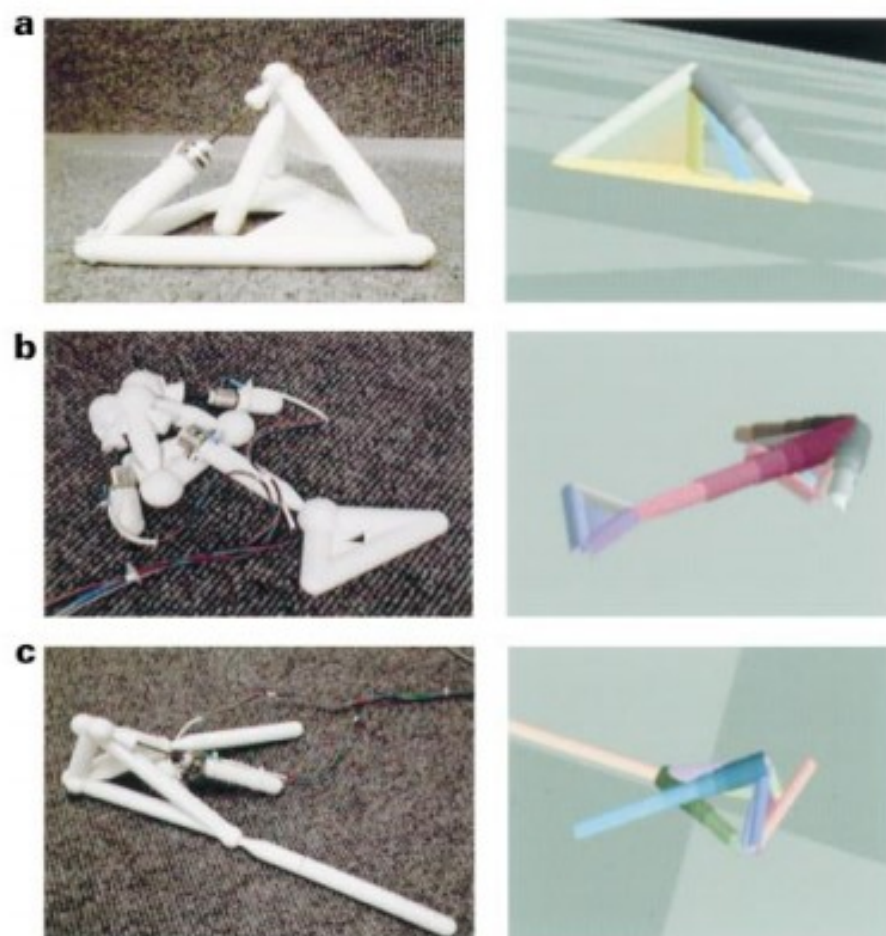
**Figure 4** Physical embodiment process. **a**, Automatically 'fleshed' joints in virtual space; **b**, a physical replication process in a rapid prototyping machine that builds the three-dimensional morphology layer after layer; **c**, pre-assembled body in mid print with discardable support structure; **d**, a close-up image of a joint printed as a single unit. The ball is printed inside the socket.





FAB@HOME





**Figure 5** Three resulting robots. Real robots (left); simulated robots (right). **a**, A tetrahedral mechanism that produces hinge-like motion and advances by pushing the central bar against the floor. **b**, This surprisingly symmetric machine uses a seven-neuron network to drive the centre actuator in perfect anti-phase with the two synchronized side limb actuators. While the upper two limbs push, the central body is retracted, and vice versa. **c**, This mechanism has an elevated body, from which it pushes an actuator down directly onto the floor to create ratcheting motion. It has a few redundant bars dragged on the floor, which might be contributing to its stability. Print times are 22, 12 and 18 hours, respectively. These machines perform in reality in the same way that they perform in simulation. Motion videos of these robots and others are available: see Supplementary Information.

**Table 1 Results**

| Machine               | Distance travelled (cm) |           |
|-----------------------|-------------------------|-----------|
|                       | Virtual                 | Physical  |
| Tetrahedron (Fig. 5a) | 38.5                    | 38.4 (35) |
| Arrow (Fig. 5b)       | 59.6                    | 22.5 (18) |
| Pusher (Fig. 5c)      | 85.1                    | 23.4 (15) |

Comparison of the performance of physical machines versus their virtual origin. Values are the net distance that the centre of mass of each machine travelled over 12 cycles of neural network. Distances given in the column headed 'physical' are compensated for scale reduction (actual distance is shown in parentheses). The mismatch in the last two rows is primarily due to the slipping of limbs on the surface.



# **Golem Project**

**Lipson & Pollack, *Nature* 2000**

# **Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers**

Sylvain Koos  
ISIR, CNRS UMR 7222  
Univ. Pierre et Marie Curie  
F-75005, Paris, France  
[koos@isir.upmc.fr](mailto:koos@isir.upmc.fr)

Jean-Baptiste Mouret  
ISIR, CNRS UMR 7222  
Univ. Pierre et Marie Curie  
F-75005, Paris, France  
[mouret@isir.upmc.fr](mailto:mouret@isir.upmc.fr)

Stéphane Doncieux  
ISIR, CNRS UMR 7222  
Univ. Pierre et Marie Curie  
F-75005, Paris, France  
[doncieux@isir.upmc.fr](mailto:doncieux@isir.upmc.fr)

This paper introduces an alternative method based on an evaluation of the *transferability*. A controller is said transferable if the corresponding trajectories of the robot (expressed in a relevant state space) in simulation and in reality are quantitatively similar. Based on this definition, we propose a new evolutionary approach that aims to:

- find controllers that are both relevant for a given task and transferable from simulation to reality;
- minimize the number of transfers by relying at most on the simulation and only conducting a few experiments on the real robot during the evolutionary run.

Solutions that behave at best in simulation frequently exploit bugs or badly modeled phenomena, making them not transferable. Transferability and efficiency therefore appear to be conflicting objectives. In order to look for relevant trade-off solutions, we propose to optimize solutions with a Pareto-based Multi-Objective Evolutionary Algorithm (MOEA) in which two objectives are defined: a task-dependent fitness only computed in simulation and a transferability objective.



### *Behavioral features and distance.*

For each controller evaluated in simulation, we assume that its corresponding behavior can be summed up by  $n$  values, called behavioral features. Once computed, these features allow to define a *behavioral distance* between individuals. Let  $\mathbf{b}^{(1)}$  and  $\mathbf{b}^{(2)}$  be the vectors of  $n$  behavioral features corresponding to the controllers  $c^{(1)}$  and  $c^{(2)}$ , the behavioral distance *b\_dist* between these controllers is:

$$b\_dist(c^{(1)}, c^{(2)}) = \|\mathbf{b}^{(1)} - \mathbf{b}^{(2)}\|^2$$

This behavioral distance allows to compare controllers in a simple and fast manner without any dependence on controllers' genotype/phenotype or assumption about it.

### *Behavioral diversity.*

To quantify the diversity of a controller from the already transferred ones, we define a behavioral diversity value as follows<sup>1</sup>. Let  $C_T$  be the set of the already transferred controllers and *b\_dist* the behavioral distance, the behavioral diversity value *diversity*( $c$ ) for a given controller  $c$  is:

$$diversity(c) = \min_{c_i \in C_T} b\_dist(c, c_i)$$

### *STR disparity.*

To estimate controller's transferability, an exact simulation-to-reality disparity value is computed by transferring a controller and comparing the corresponding real and simulated behaviors of the robot. Let us assume that some controllers have already been transferred onto the real robot, the behavioral distance defined above allows to compute an *approximated STR disparity* value for any controller  $c$ . Let  $C_T$  be the set of the already transferred controllers and  $D^*(c_i)$  the exact STR disparity value corresponding to the controller  $c_i \in C_T$ , the approximated STR disparity  $\hat{D}$  of  $c$  is:

$$\hat{D}(c) = \frac{1}{N} \sum_{c_i \in C_T} \frac{D^*(c_i)}{b\_dist(c, c_i)},$$
$$\text{with } N = \sum_{c_i \in C_T} \frac{1}{b\_dist(c, c_i)}$$

### *Evaluation objectives.*

The evaluation process only takes place in simulation. Each controller is evaluated by 2 main objectives in a multi-objective manner:

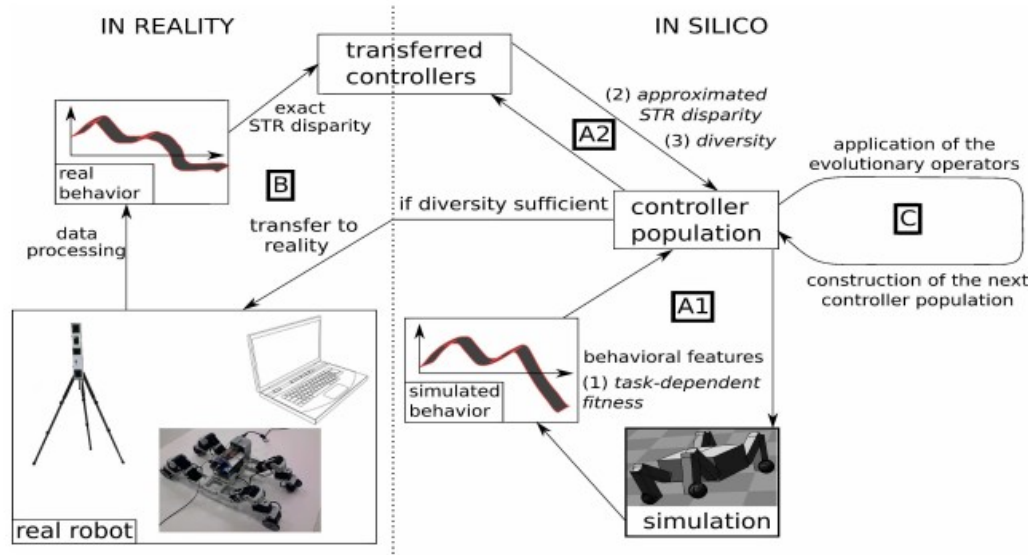
1. the task-dependent fitness, to find good controllers;
2. the corresponding approximated STR disparity, to find transferable controllers.

Moreover, in order to minimize the number of transfers we want to efficiently explore the controller state space. Exploration can be improved by maintaining behavioral diversity among the population with the help of a third objective [16]:

3. the behavioral diversity value defined above.

Such a diversity objective looks for solutions that show the more different behaviors from those of the already transferred controllers and ensures that any new experiment is meaningful.





**Figure 2: Steps of the proposed algorithm at each generation – A1.** The behavioral features and the task-dependent fitness are evaluated for each controller in simulation. **A2.** The behavioral features of a given controller allow to compute the corresponding approximated STR disparity along with the diversity value based on behavioral distances to the already transferred controllers. **B.** If this behavioral diversity value is high enough for some controllers, one among them is transferred onto the real system and the corresponding exact STR disparity is computed by comparing the observed simulated and real behaviors of the robot. **C.** The evolutionary operators are applied to controllers and the selection step builds the next population.

### 3.2 Algorithm outline

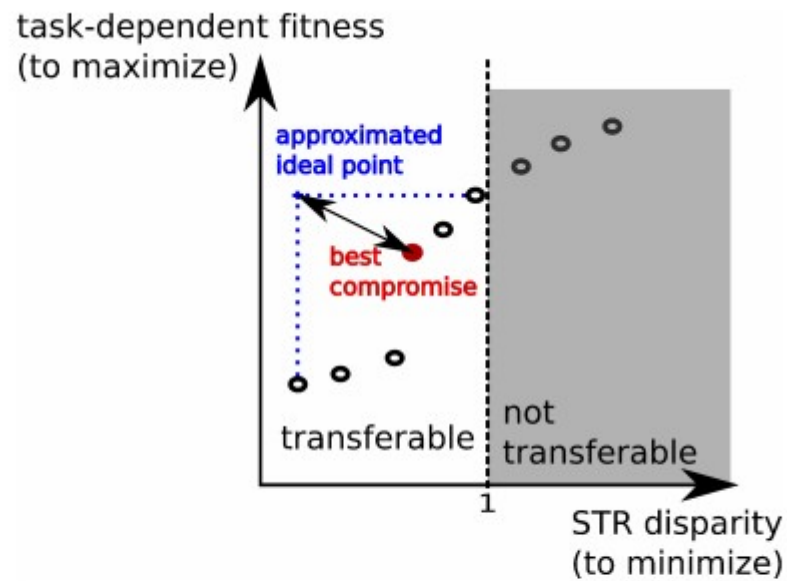
To compute the approximated STR disparity values at the beginning of a run, we assume that a controller  $c_0$  has already been transferred onto the real system. The corresponding exact STR disparity  $D^*(c_0)$  and its behavioral features are then available and an approximated STR disparity value can be computed for each controller.

Each generation of the algorithm takes place as follows (Figure 2):

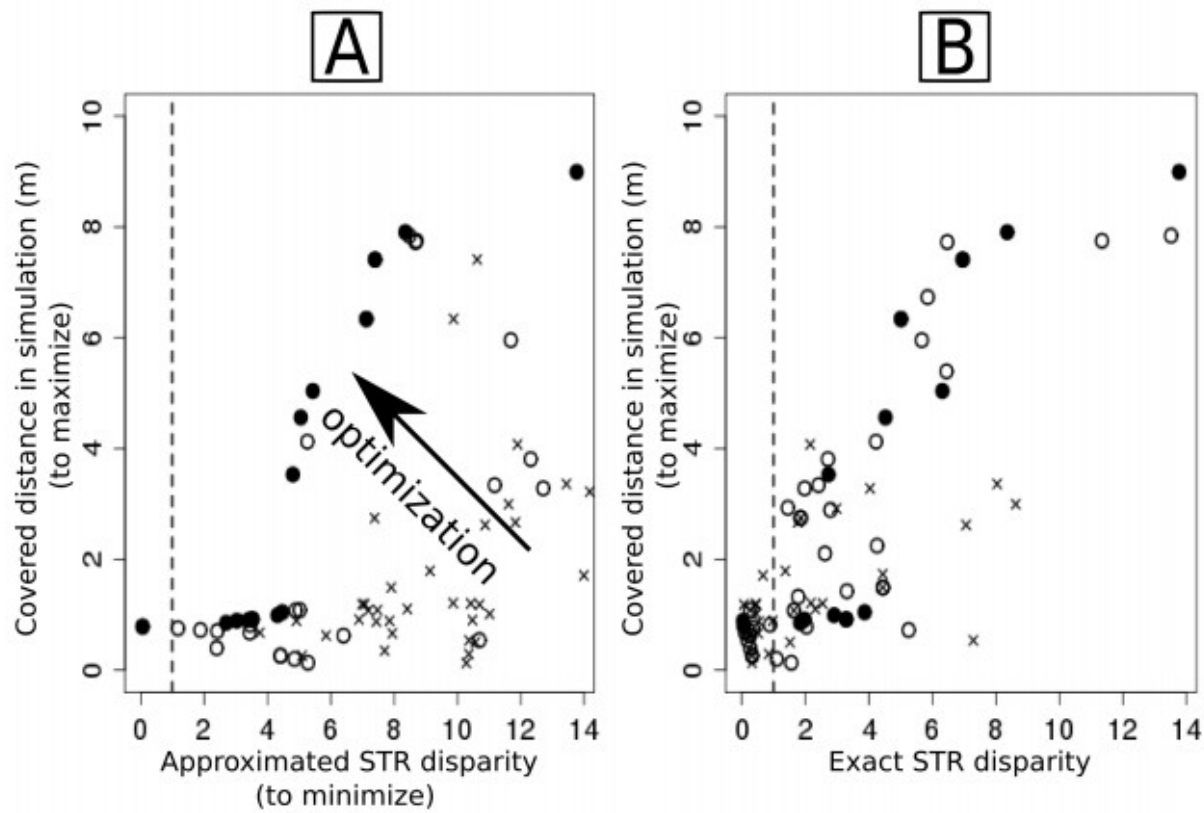
- A. evaluation of the controllers:
  - A1. computation of the task-dependent fitness objective and the behavioral features;
  - A2. evaluation of the 2 other objectives in relation to the already transferred controllers (approximated STR disparity and diversity objective);
- B. if some controllers have a high enough diversity, one among them is transferred onto the real system;
- C. application of evolutionary operators and generation of the next population.

The transfer step B occurs at each generation: once all controllers are evaluated, at most one controller from the population is transferred onto the real system.



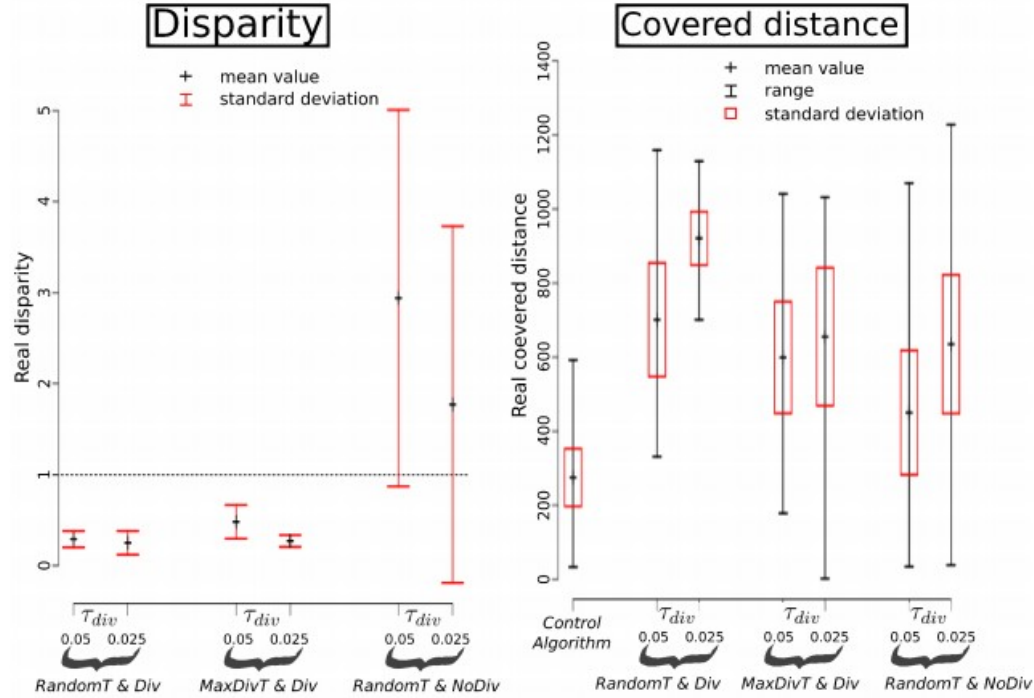


**Figure 3:** Illustration of the best compromise definition on a non-dominated set.



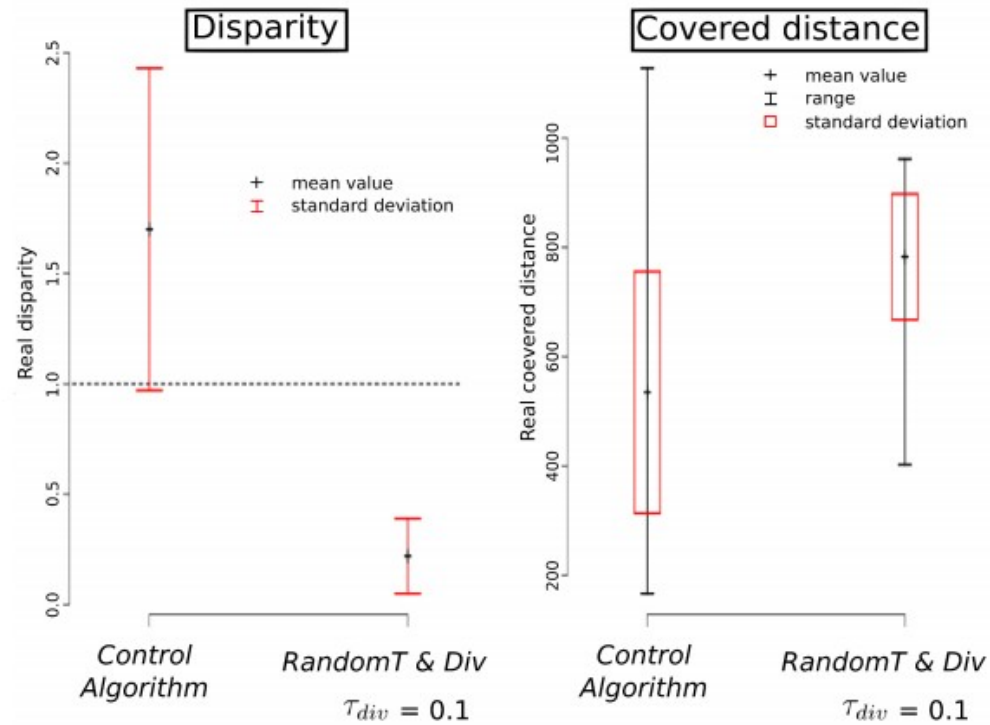
**Figure 5: Graph A:** non-dominated individuals (circle) along with last population (cross) obtained with a typical run in setup  $Exp_1$  (Proposed Algorithm, variant *RandomT* & *Div*) in terms of covered distance in the simple simulation and approximated disparity. The black circles are the non-dominated set without taking into account the diversity objective. **Graph B:** same individuals (setup  $Exp_1$ ) expressed in terms of covered distance in the simple simulation and exact disparity.

## Exp1: simulation to simulation



**Figure 6: Results for setup  $Exp_1$ : exact disparity (left, except for Control Algorithm:  $\overline{D}^* = 55.51, \sigma_{D^*} = 26.41$ ) and covered distance (mm, right) in accurate simulation of the best solutions obtained with each algorithm. The diversity threshold values  $\tau_{div}$  are written above the variant names. All variants behave better than the Control Algorithm both in terms of disparity (Student T-test p-values  $< 10^{-4}$ ) and distance (p-values  $< 10^{-2}$ ) except for *RandomT & NoDiv* variant (p-value =  $8.01 \cdot 10^{-2}$ ). Moreover, for *RandomT & Div* and *MaxDivT & Div* variants, the disparities are clearly lower than 1: the found best solutions show good transferability properties.**

## Exp2: simulation to real robot



**Figure 7: Results for setup *Exp2*: exact disparity (left) and covered distance in reality (mm, right) of the best solutions obtained with each algorithm. Due to the few repetitions (5), it is hazardous to compute any relevant statistical significance. However, the *RandomT & Div* variant behaves clearly better than the Control Algorithm with disparities lower than 1 and finds more efficient controllers on average regarding the covered distance objective.**



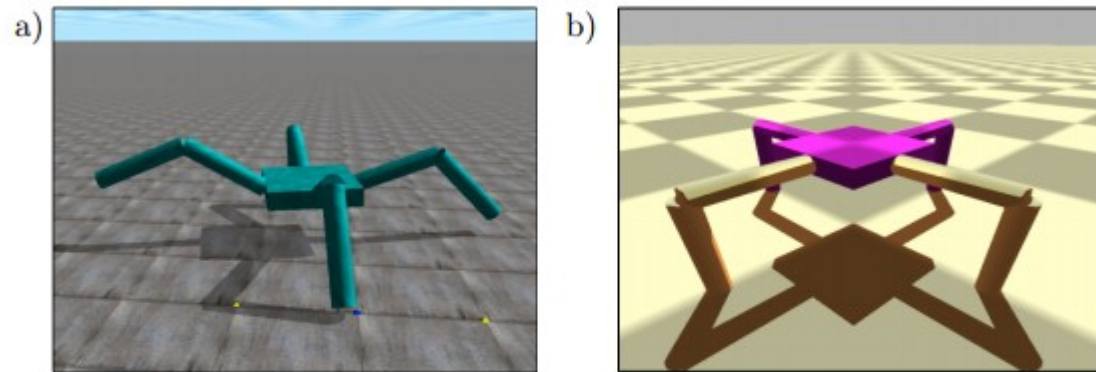
# **Not All Physics Simulators Can Be Wrong in the Same Way**

Shane Celis

Morphology, Evolution & Cognition Laboratory  
Vermont Complex Systems Center  
College of Engineering and Mathematical  
Sciences  
University of Vermont  
scelis@uvm.edu

Josh Bongard

Morphology, Evolution & Cognition Laboratory  
Vermont Complex Systems Center  
College of Engineering and Mathematical  
Sciences  
University of Vermont  
jbongard@uvm.edu



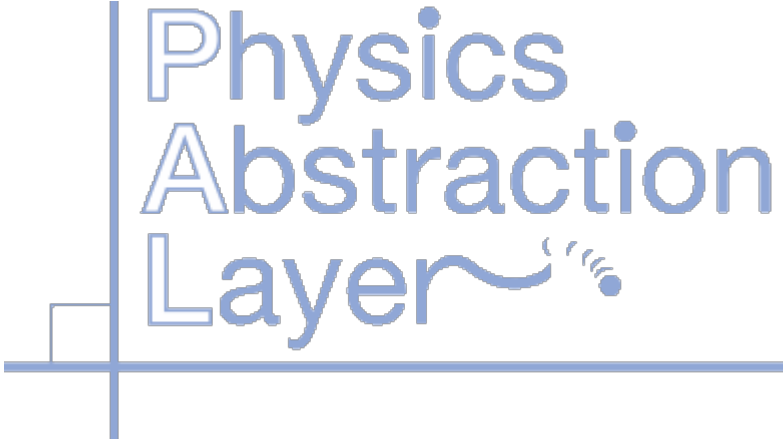
**Figure 1: Quadruped in a) Open Dynamics Engine and b) Bullet Physics Library**

The robot is evaluated in two physics simulators: 1) the Open Dynamics Engine (ODE) and 2) the Bullet Physics Library. The physics simulators use a fixed time step of 0.01 seconds. The robots are evaluated for 10 simulated seconds.

The distance traveled in the  $x$ - $y$  ground plane by ODE and Bullet is denoted by  $d_1$  and  $d_2$  respectively, measured in meters. Three experiments are conducted based on these measurements. Experiment A maximizes  $d_1$  and minimizes  $|\Delta d| = |d_1 - d_2|$ . Experiment B maximizes  $d_1$  and maximizes  $|\Delta d|$ . Experiment C, the control experiment, maximizes  $d_1$  only. Twenty independent trials of each experiment are performed using the multi-objective optimization algorithm NSGA-II[2] with a population of 20 and 250 generations.

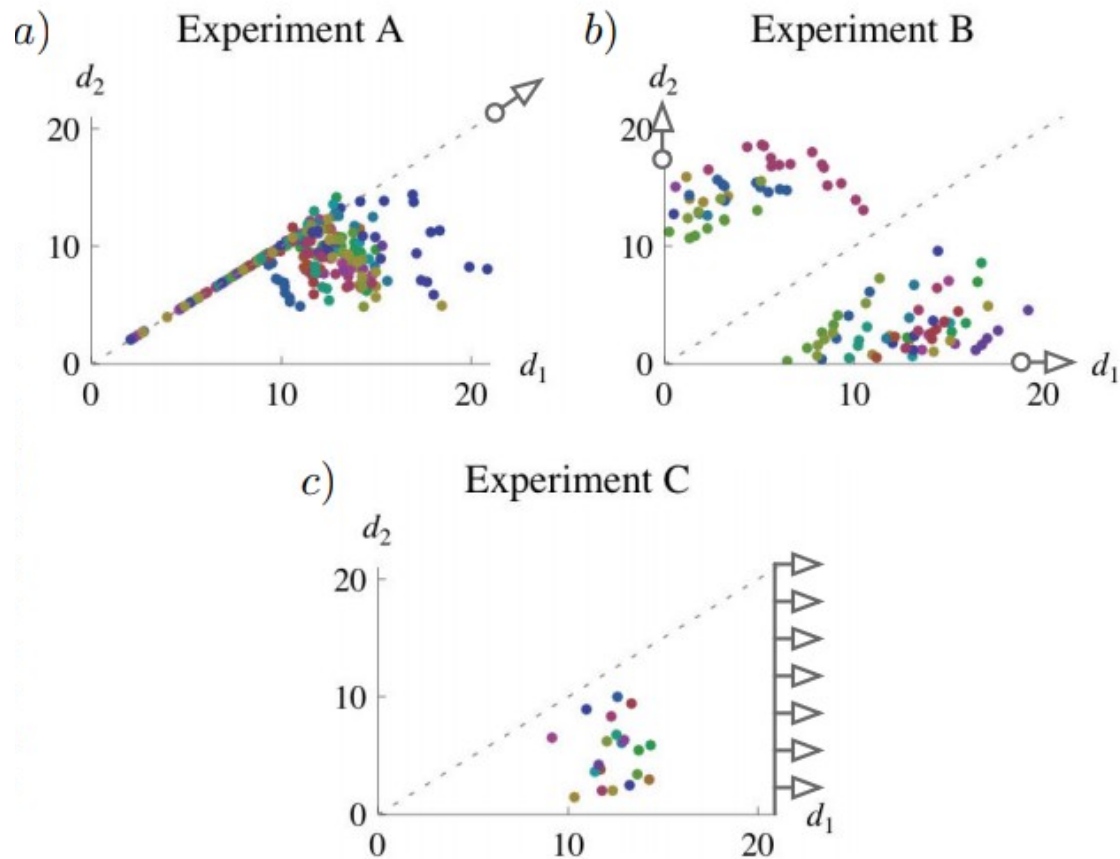


# Physics Abstraction Layer



PAL supports a large number of [physics engines](#). It provides a unique interface for:

- Physics Engines
  - [Box2D](#) (experimental)
  - [Bullet](#)
  - [Dynamechs](#)(deprecated)
  - [Havok](#) (experimental)
  - [IBDS](#) (experimental)
  - [JigLib](#)
  - [Meqen](#)(deprecated)
  - [Newton](#)
  - [ODE](#)
  - [OpenTissue](#) (experimental)
  - [PhysX](#) (a.k.a Novodex, Ageia PhysX, nVidia PhysX)
  - [Simple Physics Engine](#) (experimental)
  - [Tokamak](#)
  - [TrueAxis](#)



**Figure 2:** The points represent the distances achieved by non-dominated controllers in both physics simulators. The color of each point represents which of the twenty trials produced it: same color, same trial. The dashed line represents the case where the distances are the same, i.e.  $\Delta d = 0$ . The arrows represent the direction of optimization for each experiment. Experiment B shows that a controller can be found that performs well in one simulator while doing poorly in the other. Experiment C shows that  $d_2 < d_1$  in all cases.