Evol. Intel. (2012) 5:45–56
DOI 10.1007/s12065-011-0067-y

SHORT NOTE

Constructing controllers for physical multilegged robots using the ENSO neuroevolution approach

Vinod K. Valsalam · Jonathan Hiller · Robert MacCurdy · Hod Lipson · Risto Miikkulainen

Received: 5 June 2011 / Revised: 3 November 2011 / Accepted: 5 December 2011 / Published online: 6 January 2012

Abstract Evolving controllers for multilegged robots in simulation is convenient and flexible, making it possible to prototype ideas rapidly. However, transferring the resulting controllers to physical robots is challenging because it is difficult to simulate real-world complexities with sufficient accuracy. This paper bridges this gap by utilizing the Evolution of Network Symmetry and mOdularity (ENSO) approach to evolve modular neural network controllers that are robust to discrepancies between simulation and reality. This approach was evaluated by building a physical quadruped robot and by evolving controllers for it in simulation. An approximate model of the robot and its environment was built in a physical simulation and uncertainties in the real world were modeled as noise. The resulting controllers produced well-synchronized trot gaits when they were transferred to the physical robot, even on different walking surfaces. In contrast to a hand-designed PID controller, the evolved controllers also generalized well to changes in experimental conditions such as loss of voltage and were more robust against faults such as loss of a leg, making them strong candidates for real-world applications.

Keywords Modular controller · Symmetry · Neuroevolution · Physical simulation · Quadruped robot · Gait

V. K. Valsalam (✉) · R. Miikkulainen
Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, USA
e-mail: vkv@cs.utexas.edu

J. Hiller · R. MacCurdy · H. Lipson
Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA

1 Introduction

Imagine having to design a multilegged robot to explore Mars. Such a robot could navigate the rugged terrains of Mars better than a wheeled robot, but it is more difficult to control. The controller must coordinate the robot's legs properly, generating robust gaits to negotiate different terrains effectively while maintaining stability. Moreover, the robot should be robust to different environmental conditions, wear and tear, and even failure like losing one or more legs, to reliably complete its mission.

Research on designing controllers for multilegged robots began in the 1980s [1, 6, 29] and continues actively both in academia and industry [3, 8, 15, 19]. The controllers for state-of-the-art robots are often designed by hand, requiring extensive analysis of the sensor-motor systems and body-limb dynamics [3, 8, 15, 19, 29]. This process is generally difficult and brittle because it is hard to anticipate all operating conditions.

Therefore, automatic design methods utilizing learning techniques such as evolution are a promising alternative [2, 4, 14, 35]. These methods typically evaluate controller fitness by simulating the physical behavior of robots and their environments. Such experiments are useful because they allow testing a range of conditions quickly and effectively without damaging expensive hardware components of the robot. However, simulation may not always produce accurate enough results, making it difficult to transfer the evolved controller to the physical robot [22, 23].

This paper evaluates the hypothesis that controllers evolved in an approximate simulation can be transferred successfully to the physical robot if the evolved controllers are robust enough. An approximate simulation is first made by modeling the weight distribution of the robot and the uncertainties in its sensors and actuators. An approach

called Evolution of Network Symmetry and mOdularity [ENSO; 33, 35, 36] is then utilized to evolve controllers that transfer robustly from simulation to reality. This method was demonstrated on a physical quadruped robot that was designed and built using rapid prototyping technology. The evolved controllers produced the same gaits both in simulation and on the physical robot. Moreover, they generalized better to different experimental conditions and demonstrated better fault tolerance than a hand-designed PID controller, suggesting that such evolved controllers are useful for real-world applications.

2 Background

This section reviews prior work on evolving controllers for physical robots and summarizes the ENSO approach that this paper utilizes to evolve controllers.

2.1 Evolving controllers for physical robots

It is possible to evolve controllers by evaluating their fitness directly on the real robot instead of in simulation [11, 16, 20, 37, 40]. However, performing thousands of such fitness evaluations in hardware may be impractical for several reasons [23]. First, hardware evaluations are slow, resulting in long evolutionary run times. Second, they cause wear and tear on the robot, making hardware failures likely and user intervention to repair them necessary. Third, the controllers created by evolution through random variations may produce abnormal actuator signals that can crash or damage the robot.

Therefore, a good alternative is to evaluate controller fitness in simulation and then transfer only the final, evolved controller to the physical robot. However, transferring such controllers evolved in simulation to the physical robot is challenging [7, 22, 23]. The main reason is that it is difficult to simulate physical properties such as friction and sensor and actuator characteristics with high enough fidelity to reproduce the simulated behaviors on real robots. In order to address this issue, researchers have developed several methods that improve the results of evolution in simulation by performing only a few experiments on the real robot.

A straightforward method is to fine-tune the controller behaviors by continuing evolution on the real robot for a few more generations [24, 25]. However, this method may be ineffective in correcting behaviors that have evolved to exploit flaws in the simulation. A better alternative is to make such behaviors less likely to evolve by incorporating transfer experiments from the beginning of evolution, e.g. by utilizing a multi-objective evolutionary algorithm that optimizes both a task-dependent controller fitness as well as a measure of how well the controller transfers from simulation to reality [21]. In any given generation, this method chooses at most one controller based on behavioral diversity to be evaluated on the real robot, requiring only a small number of hardware evaluations.

Other methods utilize the information they gather from a few experiments on the real robot to build a more realistic simulator, typically in one of two ways: (1) Experiments are performed on the real robot before running evolution to collect samples of the real world by recording sensor activations [24, 25]. When controllers are evaluated later during evolution, these samples are utilized to set the simulated sensor activations accurately. (2) Experiments are performed on the real robot during evolution to co-evolve the simulator and the controller, making an initially crude simulation more and more accurate [5, 7, 38].

In contrast to the above methods, this paper proposes to bridge the simulation-reality gap simply by creating controllers that are robust to small discrepancies between simulation and reality. It utilizes the ENSO approach [33, 35, 36], which evolves controllers modeled as coupled cell systems to provide theoretical guarantees of robustness. Therefore, simulation accuracies sufficient for ENSO can be obtained easily by approximating the morphology and mass distribution of the real robot with cylindrical and rectangular blocks, without performing any hardware experiments either before or during evolution (Sect. 3.2).

Evolving controllers in an accurate enough simulation is often insufficient to transfer them successfully to the real world because of uncertainties in sensor activations and actuator responses. However, past work has shown that evolution can adapt controllers to such uncertainties by modeling them as noise in the simulation [13, 17, 18, 24]. The same idea is utilized in this paper, focusing on uncertainties in how the motors respond to control signals. The resulting controllers transfer successfully to the physical robot, producing the same behaviors both in simulation and on the physical robot. The ENSO approach that is used to evolve these controllers is discussed next.

2.2 The ENSO approach

The controller for a multilegged robot can be implemented as a system of interconnected neural network modules, each controlling a different leg [1, 34]. Some of these modules and interconnections may be identical, resulting in symmetries, i.e. permutations of the modules that leave their interconnection graph invariant (Fig. 1). Symmetries can be analyzed using group theory to show that they determine the type of gaits that the controller can produce [9]. However, designing the appropriate symmetries by hand is often difficult and may even be infeasible in the general case [35]. Moreover, the parameters of the neural

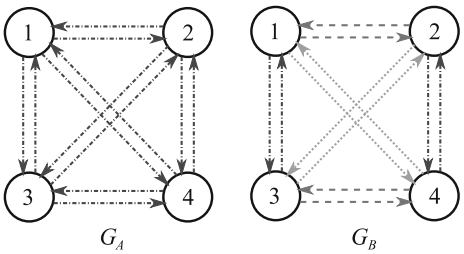


Fig. 1 Example symmetries of a modular neural network controller for a quadruped robot. The four controller modules and their interconnections can be represented as the vertices and edges of a *colored graph*. Vertices and edges of the same color (indicated by *line* style) are identical, each *color* representing a particular combination of neural network parameters. A graph symmetry is any permutation of vertices under which the edge colors remain the same. Both graphs in this figure have vertices of the same color. Moreover, all edges of graph G_A have the same color, while edges of graph G_B have different colors. Therefore, any permutation of the vertices of graph G_A is a symmetry. In contrast, only the permutations (1 2)(3 4) and (1 3)(2 4), and their compositions are symmetries of graph G_B. The set of all symmetries of a graph forms a group. For example, the symmetry group of graph G_A is the *symmetric group* $\mathcal{S}_4$ and that of the less symmetric graph G_B is a subgroup of $\mathcal{S}_4$ called the *dihedral group* $\mathcal{D}_2$. The symmetry group of the controller determines the type of gaits that it can produce [9]

network have to be optimized together with its symmetries. The ENSO approach was designed to evolve solutions to such problems automatically [33, 35, 36], and it is therefore utilized in this paper.

ENSO initializes evolution with a population of maximally symmetric solutions (Fig. 2): They have the simplest possible structure, consisting of identical modules and interconnections. This structure is represented by a colored graph with identical vertices and edges. During evolution, group-theoretic mutations [33, 35, 36] break their symmetries systematically, thus exploring less symmetric, more complex solutions with different types of modules and interconnections. Each such mutation creates a new graph with more colors such that its symmetry group is a random maximal subgroup of the original symmetry group.

ENSO organizes the colors created by successive symmetry (color) mutations as a tree. Each such tree is a genotype for evolution. The leaf nodes of this tree represent the colors of identical vertices and edges of the phenotype graph, which in turn represents a neural network (Fig. 2a). In particular, each leaf node specifies which vertices or edges have that color and stores the corresponding set of neural network parameters, i.e. the biases and connection weights of the module network (for vertex color) or the connection weights between modules (for edge color).

Symmetry mutations produce structural changes in the phenotype graph by partitioning the vertex or edge set of leaf nodes and creating a new child color for each part of the partition (Fig. 2b). Structural changes are also produced by enabling or disabling edges of the same color using edge-toggle mutations. Interleaving these two types of structural mutations with parameter mutations optimizes the modules and interconnections of simpler solutions first and elaborates on them to create more complex solutions only when necessary. Moreover, the systematic, symmetry-breaking mutations constrain the search to promising symmetries, making evolution more effective than mutating symmetry unsystematically [33, 35, 36].

These features make it possible for ENSO to evolve complex modular systems such as the quadruped controller illustrated in Fig. 3. All four modules of this controller utilize the same two-layered neural network architecture (Fig. 3a). Each module's input is the joint angle of the leg it controls. It can be represented by the angle itself, or by the sine and cosine of the angle; the sine and cosine are actually more robust (because they are continuous), and were used to obtain the results discussed in Sect. 4. The module's output is the desired angular velocity of that leg. The full controller network is obtained by connecting the four modules to each other such that each module receives input from all the other modules (Fig. 3b).

The phenotype graph representing the symmetries of the controller network also represents a central pattern generator, or coupled cell system, i.e. a dynamical system capable of generating robust periodic oscillations for modeling quadruped gaits [9]. Therefore, ENSO can design effective controllers by evolving their symmetries in addition to their parameters, as demonstrated previously in physically realistic simulations of a quadruped [35, 36]. Symmetry evolution was especially advantageous when the appropriate symmetries are difficult to determine manually, such as on an inclined ground. In particular, ENSO evolved gaits that were significantly faster and also generalized better than those evolved with only parameter mutations on hand-designed symmetries. Moreover, the group-theoretic symmetry mutations of ENSO produced smoother and better coordinated gaits than those produced using arbitrary symmetry mutations. Since the ability to evolve such robust and regular gaits is important for real-world applications, ENSO is a promising approach for designing distributed controllers for real robots. This paper investigates this hypothesis by extending previous simulation results, as will be described next.

3 Methods

A physical quadruped robot was designed and fabricated to evaluate the ENSO approach in real-world applications. This section discusses its design, simulation, and control.

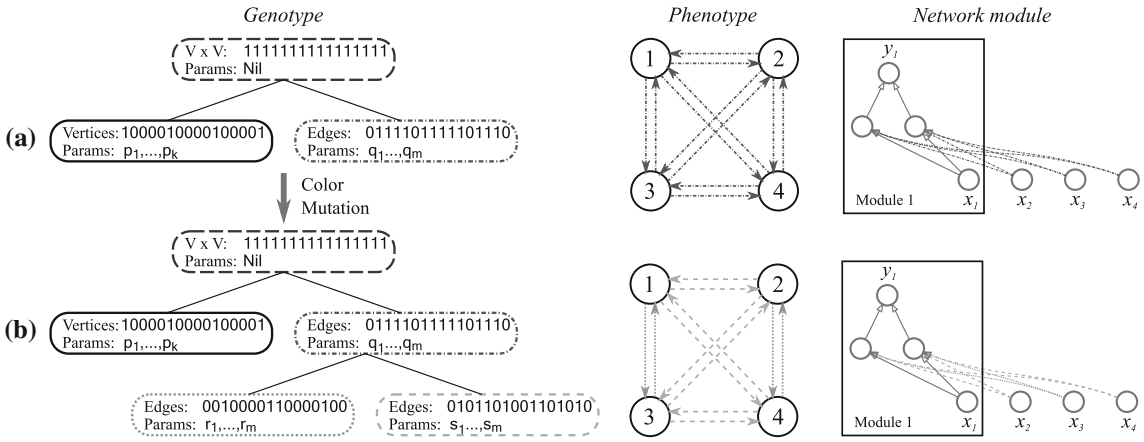


Fig. 2 Examples of genotype, phenotype, network module, and color mutation. ENSO uses a tree of colors as genotype (*left*). Each leaf of this tree has a unique color, and represents a set of vertices or edges of the phenotype graph (*middle*) that have the same parameter values. The vertices and edges of the phenotype graph represent the modules of a neural network and the connections between them (*right*). Their parameters (stored in the genotype) consist of node biases and connection weights for each module network (vertex) and weights for each connection between modules (edge). Each module has a fixed architecture with a layer of hidden nodes fully connected to its inputs and outputs. A connection from another module (not shown) is implemented by fully connecting its input layer to the hidden layer of the target module. **a** At the beginning of evolution, each genotype in the population represents a maximally symmetric phenotype graph with symmetry group $\mathcal{S}_4$. All vertices of this graph have the same color (*solid*, represented by the leaf on the left) and all its edges have the same color (alternating *dots* and *dashes*, represented by the leaf on the right), implying that all modules are identical and all connections between them are also identical. **b** A color mutation breaks the phenotype graph symmetry to $\mathcal{D}_4$, which is a maximal subgroup of $\mathcal{S}_4$. As a result, two child nodes are created for the node representing the set of edges, i.e. the set of edges is partitioned into two and each part is colored differently (*dotted* and *dashed*). Since each color is associated with a different combination of parameter values, the mutated phenotype graph represents two types of connections between network modules. Such color mutations, when combined with parameter mutations, make it possible to evolve symmetric and modular neural networks efficiently

3.1 Physical robot

The physical robot was designed with two constraints: (1) it must be similar to the simulated robots that were previously studied using ENSO [35, 36], and (2) it must be possible to prototype it quickly with parts that can be purchased or fabricated easily. While some parts of the robot such as the servo motors and the controller board to operate the motors were available commercially, other parts such as its body and legs were custom-designed and fabricated to fit the commercial parts.

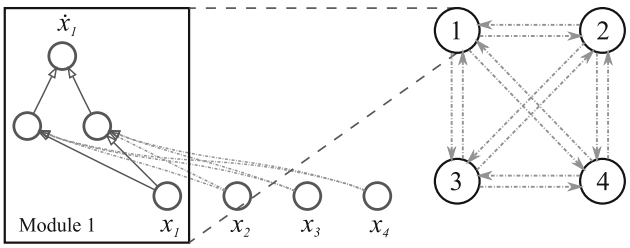
Each leg of the robot is attached to a Dynamixel AX-12+ servo motor manufactured by Robotis [30]. The AX-12+ provides angular position feedback, and can be made to rotate continuously by specifying the desired angular velocity, thus matching the inputs and outputs of the neural network controllers that ENSO evolves. The evolved controller runs on a Robotis CM-2+ microcontroller circuit board, which provides an interface to communicate with the Dynamixel motors through a daisy-chain serial connection.

The AX-12+ motors are mounted on a rectangular body using a wedge-shaped piece to tilt their axes of rotation 20° from the vertical (Fig. 4). The legs also slant 20° from their respective motor axes, making it possible for the robot to walk by rotating its legs continuously. The body, the wedge, and the leg were designed using SolidWorks [31], a program for computer aided design. The body was then cut from acrylic using a laser cutter and the wedges and the legs were fabricated in an Objet Eden 260 V [26] rapid-prototyping 3D printer. The circuit board is mounted on the top side of the body and is powered by a 12 V lithium-ion battery attached to its bottom side by Velcro.

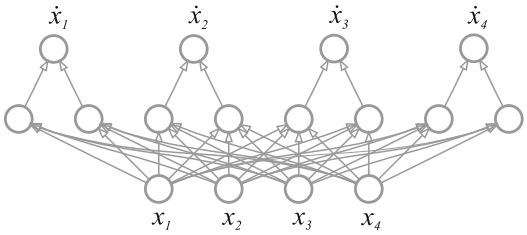
This robot was then modeled in a physics simulation for evolving neural network controllers using ENSO. The simulation utilized OPAL [28], an abstraction library on top of the Open Dynamics Engine (ODE) [27], and is described in detail next.

3.2 Simulation

The robot's legs were modeled as cylinders with capped ends, while its body was assembled from several rectangular boxes that approximate different body parts (Fig. 5). These cylinders and boxes have the same dimensions and the same relative angles as the corresponding parts in the physical robot. The leg angles used as controller input are measured from the same vertical leg positions. Moreover, densities were assigned to the parts such that they have the same mass in both simulation and the real robot. As a result, the simulation utilizes the same approximate morphology and mass distribution as the physical robot.



(a) Network module and the initial phenotype graph



(b) Full controller network consisting of four modules

Fig. 3 Modular controller network for a quadruped robot. The input to each module is the angle (or its sine and cosine) of the leg it controls, and the output is the desired angular velocity of that leg. The full controller network consists of four such modules, each module receiving input from all the other modules. The symmetries of these modules and their connectivity are represented by a phenotype graph. At the beginning of evolution, this graph has identical vertices (modules) and edges (interconnections), i.e. all vertices and edges have the same combination of network parameters. ENSO discovers effective controllers by breaking symmetry to create new types of vertices and edges, and by optimizing their initially random network parameters. **a** Network module and the initial phenotype graph. **b** Full controller network consisting of four modules

In addition to the morphology, the sensor and actuator characteristics of the AX-12+ motors were also modeled. The motor can sense its angular position if it is in the [0, 300] degree range (Fig. 6). However, it does not give valid position feedback for angles between 300° and 360°. Since the neural network controllers take angular positions as inputs, the sensor reading is interpolated when the motor is in this blind zone. In fact, the sensor reading is calculated the same way for all angular positions from an estimate of the angular velocity, which gets updated only when the angle is in the valid range. Exponential smoothing is applied to this estimate to filter out noise and discontinuities caused by any discrepancy between the estimated and actual angular velocities when the motor emerges from the blind zone.

The response of the motor to the angular velocity control signals from the neural network is more difficult to model accurately. In particular, the angular velocity of the motor drifts significantly over time for a constant control signal. This stochastic drift can change the periodic trajectory of the controller, thus disrupting the gait of the robot. Such uncertainties can be handled by adding noise to the simulation, making it possible for evolution to adapt the

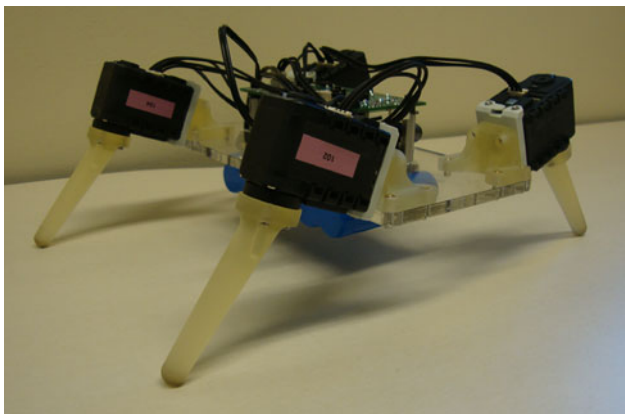


Fig. 4 Assembled physical quadruped robot. The Dynamixel AX-12+ motors rotate the legs of the robot and are mounted on the four corners of its rectangular acrylic body, which has attachment points in the middle for a future hexapod extension. The legs make 20° with the axis of rotation of their respective motors, tracing cones as they rotate. The motor axes also have a 20° sideways tilt from the vertical. As a result, rotating the legs raises and lowers them and can produce locomotion when they make contact with the ground. A control program synchronizes the rotation of the legs to produce locomotion and runs on the CM-2+ circuit board mounted on top of the body. The board is powered by a 12 V lithium-ion battery attached to the under-side of the body. Videos of experiments using this robot can be seen at the website http://nn.cs.utexas.edu/?enso-realrobots

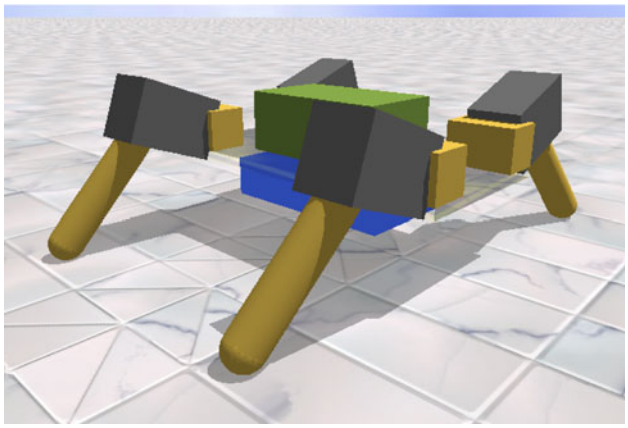


Fig. 5 Simulation model of the physical quadruped robot. This model simulates the morphology and dynamics of the physical robot in Fig. 4. The legs were modeled as capped cylinders and the other parts were approximated as *rectangular boxes* with the same dimensions. The weights of these shapes were then matched with those of the corresponding parts of the physical robot. As a result, this model represents the weight distribution of the physical robot with sufficient accuracy for simulation

controllers suitably [13, 17, 22]. Two types of Gaussian noise were added: The first type models fluctuations about the mean with standard deviation 2.5%. The second type of noise models drifts in the mean; it is therefore larger in magnitude (standard deviation 20%), but it is applied only a few times in each evaluation.

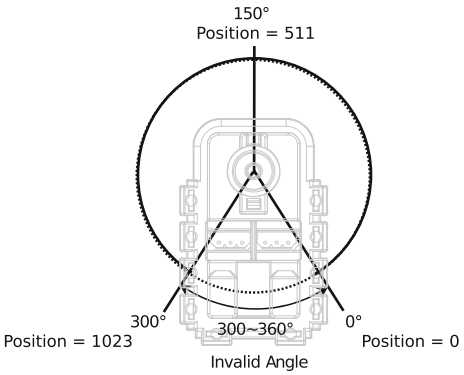


Fig. 6 Angular position sensor readings of the Dynamixel AX-12+ motor. The angular position sensor of the motor provides an integer valued reading in the range [0, 1023] when the motor is in the [0, 300] degree range; angles outside this range produce invalid sensor readings and are therefore interpolated. Reprinted with permission from Robotis [30]; annotations edited

The accuracy of the motor in representing its angular position and velocity is also included in the simulation. The motor represents both variables with an integer value in the range [0, 1023]. The precision of these variables in simulation is therefore downgraded from floating point precision to match the actual precision on this robot. Moreover, the neural network controller reads the positions and updates the velocities at the same frequency in both the simulation and the real robot.

The controller evolved in simulation in the above manner is transferred to the real robot by programming the CM-2+ circuit board with it, as described next.

3.3 Control programs

The CM-2+ circuit board contains an ATmega128 CPU, which is an 8-bit microcontroller with 128 KB of on-chip programmable flash memory. Running the programs stored in its memory can activate the motors of the robot. Therefore, the evolved neural network controller is interfaced with the motors by converting it to a C-language program and invoking it from a control loop similar to that used in simulation. This C program is then cross-compiled for the ATmega128 using the GNU compiler toolchain and downloaded to the CM-2+ board through an RS-232 serial connection.

This facility for writing control programs in C was also utilized to hand-code a baseline controller for comparing with the evolved controllers. The hand-designed controller replaces the neural network in the control loop with a PID controller that also controls the leg angular velocities utilizing feedback of leg positions. Thus, the hand-designed controller also benefits from the mechanism mentioned in Sect. 3.2 for interpolating and smoothing sensor readings of leg positions. It computes the error signals for PID control as the difference between the actual leg positions and the desired leg positions for obtaining uniform leg angular velocities.

The results of experiments comparing this controller with the evolved controllers are described in the next section.

4 Results

The gaits produced by the evolved and hand-designed controllers were evaluated for walking on flat ground (1) when all four legs of the robot are functional and (2) when one leg is disabled to simulate a real-world motor failure. The controllers produced in the first experiment were evaluated further for generalization by reducing the maximum speed of the motors and by initializing one of the legs with a large error. In each experiment, generalization was also tested by placing the robot on different surfaces. These experiments and the resulting gaits are discussed in the following subsections. Videos of example gaits can be seen at http://nn.cs.utexas.edu/?enso-realrobots.

4.1 Experimental setup

In each evolutionary run, the initial population of controller networks had the architecture depicted in Fig. 3. Their connection weights were set randomly within $(-2, 2)$, neuron biases to 0, and neuron sigmoid slopes to 1. During evolution, these parameters were mutated with Gaussian perturbations (with $\sigma = 0.2$) acting with a specified probability (0.5). All edges were enabled in the phenotype graphs of the initial controllers, and mutations toggled them with a specified probability (0.1). In each generation, an offspring was created by first selecting a parent in a two-way tournament, and then applying either a parameter mutation, an edge-toggle mutation, or a symmetry mutation. Parameter mutations were 100 times more likely and edge-toggle mutations were ten times more likely than symmetry mutations. Each symmetry mutation created five offspring, all having the same symmetry, and their newly created parameters were initialized randomly. In addition, the network with the best fitness was copied without change to the next generation. A population size of 200 was used in all experiments. These particular settings were found to work well empirically and small variations did not produce a noticeable impact on the evolved results.

Each network was evaluated in a simulation (Sect. 3.2) in which the network controlled the locomotion of a robot. When the robot was initially placed in the simulation environment, its longitudinal and lateral axes were aligned with the coordinate directions of the ground plane. The simulation was then carried out for one minute of simulated

time with step size 0.01s, after which the fitness of the controller network was calculated as the distance traveled by the robot along the longitudinal axis. This fitness measure ensured that the controller networks were rewarded for moving forward in a straight line.

For all experiments, evolution was run for 500 generations and repeated 10 times, each time with a different random number seed. The following subsections discuss the results of each experiment in detail.

4.2 All legs enabled

Figure 7 shows the phenotype graph of a champion neural network controller that ENSO evolved by utilizing the simulation model described in Sect. 3.2. Its symmetry group is similar to that of the graphs that ENSO evolved for a simpler simulated quadruped in prior work [35]. Therefore, it generates a similar, well-synchronized trot gait (Fig. 8a), thus extending the previous results to a more complex and realistic robot model.

Moreover, transferring this controller to the physical robot reproduces the same gait accurately (Fig. 8b). The robot walks smoothly in a straight line even on very different surfaces such as linoleum and carpet, demonstrating that ENSO can evolve such robust controllers that transfer successfully from simulation to real robots.

The hand-designed PID controller was also tested on the real robot with a reference waveform for a trot gait having approximately the same period as the evolved controller (Fig. 9). The legs are first positioned on the reference waveform so that there is no error for the controller to correct when it starts. Thereafter, the PID mechanism of the controller corrects small errors by speeding up or slowing down the legs to keep them aligned with the reference waveform. As a result, it produces a trot gait similar to the evolved controller. However, it is not as robust and does not generalize as well as the evolved controller, as demonstrated by the experiments discussed next.

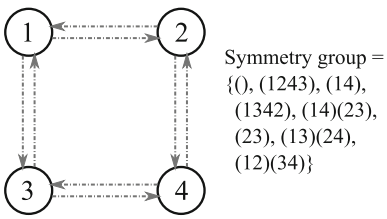
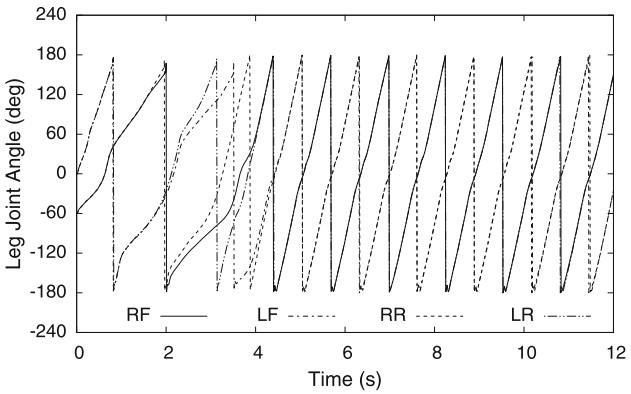
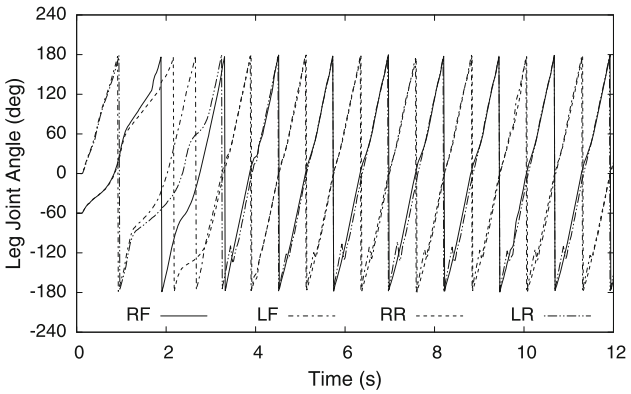


Fig. 7 Phenotype graph of a champion neural network controller evolved by ENSO. This graph has symmetry group similar to the graphs that ENSO evolved for a simpler simulated quadruped in prior work [35]. As a result, it generates a similarly well-coordinated trot gait and it transfers well from simulation to the real robot (Fig. 8)



(a) Simulated robot



(b) Real robot

Fig. 8 A trot gait evolved in simulation and transferred to the real robot. The *plots* show the four leg angles of the robot in the first 12 s. They were produced by the controller with the phenotype graph illustrated in Fig. 7. In both *plots*, after the controller reaches a steady state in about four seconds, it maintains synchrony and phase relations between the legs, producing a well-coordinated trot gait. This gait works on various surfaces robustly. Both plots are very similar, indicating that the controller produces the same walking behavior in both the simulated model and the real robot. **a** Simulated robot. **b** Real robot

4.3 Generalization to reduced motor speed

The maximum leg angular velocity that the motors can produce depends on the conditions in which the robot operates. For example, it decreases when the input voltage to the motor decreases as a result of e.g. low battery charge or temperature [12, 39]. The challenge for the controller is to keep the legs synchronized and to keep the robot walking effectively even in such conditions. The hand-designed and evolved controllers were tested for their ability to generalize to such conditions by reducing the maximum angular velocity that the motors produce.

The hand-designed controller fails, losing leg synchronization, even for a small (10%) reduction in the maximum angular velocity (Fig. 10a). It fails because the legs can no longer move fast enough to keep up with the reference waveform. Slowing down the waveform can correct the

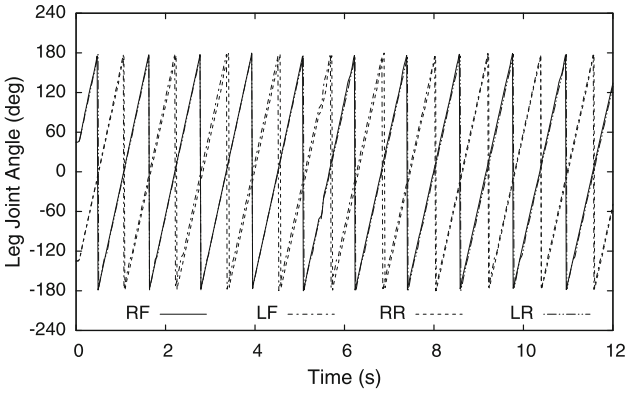


Fig. 9 Trot gait produced by the hand-designed controller for the real robot. The *plot* shows the four leg angles of the robot in the first 12 s. This controller keeps the legs synchronized with a reference waveform for a trot gait by applying PID control to correct small errors in leg positions. However, it is difficult to design such controllers by hand to work robustly in the general case

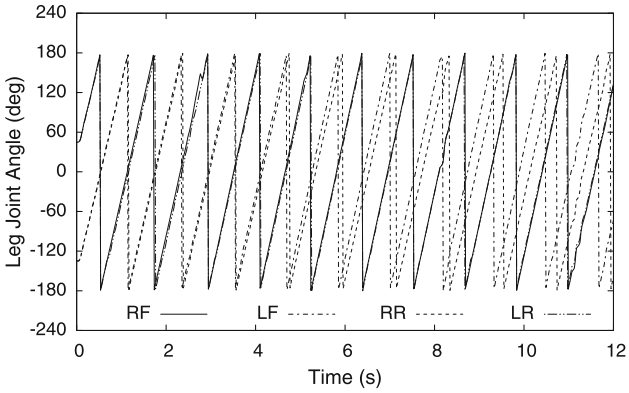
problem, but doing so in a way that produces the fastest possible gait robustly is difficult, because it requires controlling both the waveform and the leg angular velocities simultaneously. In contrast, the evolved controller continues to function robustly even when the maximum angular velocity is reduced by 60% (Fig. 10b). It achieves this robustness by slowing down the legs automatically and keeping them synchronized. Thus, the evolved controller generalizes well to a range of motor speeds, while the hand-designed controller generalizes poorly.

4.4 Generalization to different leg positions

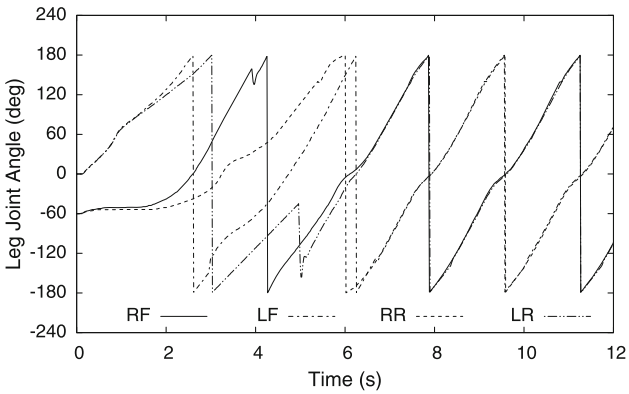
Another situation in which the hand-designed controller performs poorly is when the error between the position of a leg and the reference waveform becomes too large, which could happen e.g. when the leg is obstructed by an obstacle. The larger the error, the longer it takes the PID mechanism of the hand-designed controller to correct the error. During this time, the leg may not be synchronized well enough with the other legs to produce a good gait. The worst such behavior occurs when the error is maximum, i.e. when the leg is 180° out-of-phase with the reference waveform.

In order to evaluate robustness against such errors, the legs were first positioned such that one leg (the left-rear one) had the maximum error of 180° and the other legs had zero error. The controller was then initialized with these leg positions, making it possible to observe how quickly it corrects the error of the left-rear leg.

The hand-designed controller takes more than 30 seconds to correct this error (Fig. 11a). During this time, the angular position of the left-rear leg overshoots and undershoots the reference waveform several times, synchronizing with the right-front leg to produce the original trot gait



(a) Hand-designed controller for 10% reduction in maximum motor speed

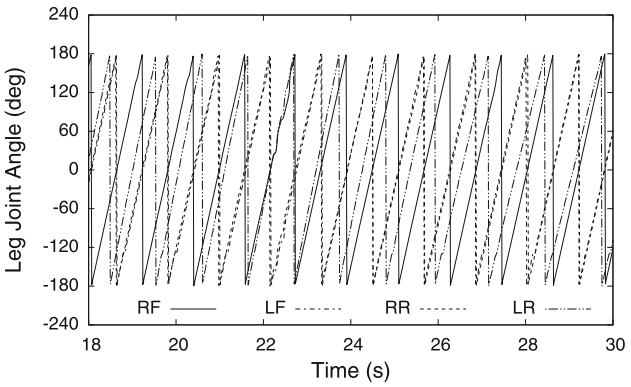


(b) Evolved controller for 60% reduction in maximum motor speed

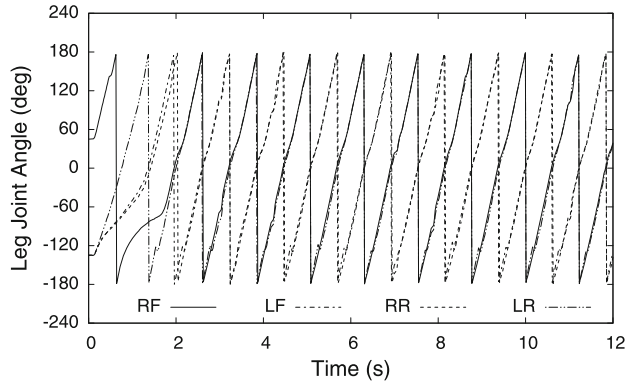
Fig. 10 Gaits produced by the hand-designed and evolved controllers on the real robot when the maximum speed of the motors is reduced. The *plots* show the four leg angles of the robot in the first 12 s. They were produced by the same hand-designed controller that produced the gait in Fig. 9 and the same evolved controller that produced the gait in Fig. 8. **a** Reducing the maximum speed of the motors even by 10% causes the hand-designed controller to lose leg synchronization quickly because it cannot keep up with the reference waveform. **b** The evolved controller maintains leg synchronization and performs robustly even when the maximum speed of the motors is reduced by 60%. It does so by simply slowing down the gait automatically. Thus the evolved controller is more general than the hand-designed controller

only gradually. Meanwhile, the other three legs that were initialized with zero error track their respective reference waveforms closely from the beginning. This behavior is the consequence of correcting the error of each leg separately without modifying the behavior of the other legs.

In contrast, the evolved controller takes only about two seconds to correct the same error (Fig. 11b). Moreover, it synchronizes the legs without producing the undesirable overshooting and undershooting oscillations (ringing) that the hand-designed controller produces. This robust behavior is possible because the control module for each leg utilizes inputs from the other legs as well. As a result, the evolved controller adjusts the behavior of all legs



(a) Hand-designed controller



(b) Evolved controller

Fig. 11 Gaits produced by the hand-designed and evolved controllers on the real robot when the left-rear leg is initialized with maximum angular position error. The *plots* show the variation of the four leg angles of the robot with time. They were produced by the same hand-designed controller that produced the gait in Fig. 9 and the same evolved controller that produced the gait in Fig. 8. In order to produce the original trot gaits, the controllers must correct the initial error by synchronizing the left-rear leg with the right-front leg. **a** The hand-designed controller adjusts only the behavior of the left-rear leg to correct the error. As a result, the left-rear leg leads and trails the right-front leg alternately, eventually synchronizing only after more than 30 s. **b** In contrast, the evolved controller adjusts the behaviors of multiple legs simultaneously, correcting the error and achieving synchronization smoothly in about 2 s. Thus the evolved controller generalizes well, while the hand-designed controller is less robust

simultaneously, bringing them into the appropriate relative phases much quicker and smoother than the hand-designed controller.

Evolution utilizes this ability of one control module to influence the behavior of the other modules in the next experiment as well, producing a straight and effective gait even when one leg is disabled.

4.5 One leg disabled

An important requirement for many robots in the real world is fault tolerance [10]. For example, hardware failures are likely in a robot operating in hostile environments or

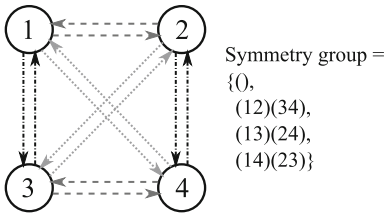


Fig. 12 Phenotype graph of a champion neural network controller evolved with the left-rear leg disabled. The symmetry group of this graph is surprising in that ENSO did not evolve a different module for the disabled leg; instead, it evolved the same module for all four legs. As a result, it produces a gait resembling trot with the disabled leg not responding (Fig. 13). ENSO adapted this gait to make the robot walk straight with only three legs

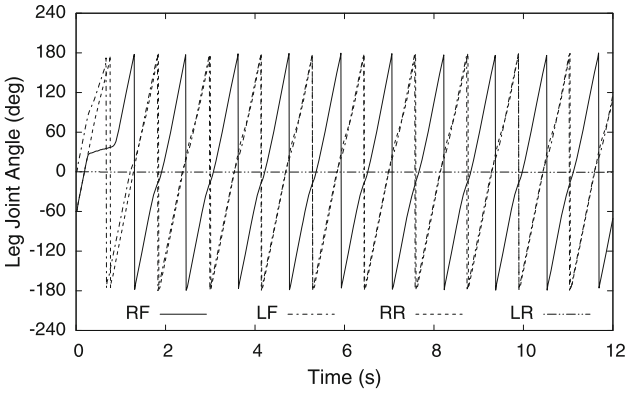
exploring another planet. In such applications, it is not always possible to replace a failed leg actuator with a new one. Walking with the same gait as before is also not an option because the asymmetric action of the remaining legs would cause the robot to curve to one side.

In such a situation, a new controller must be designed to make the robot walk effectively with its remaining legs, recovering as much performance as possible. Designing such a controller by hand is challenging; even designing the appropriate symmetry by hand is challenging. In contrast, ENSO should be able to evolve effective neural network controllers for such an asymmetric robot automatically by disabling the failed leg in simulation. The new controller can then be downloaded to the physical robot for a successful walk.

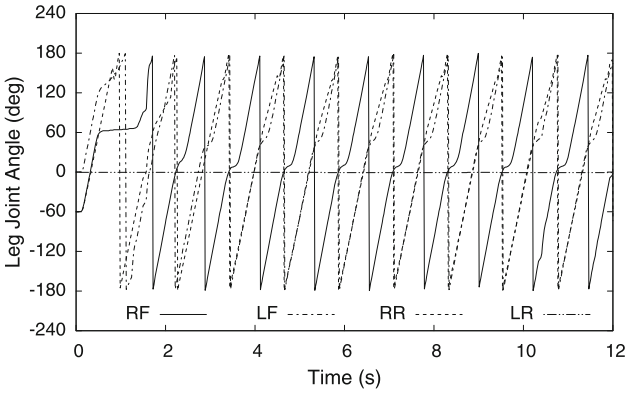
This hypothesis was tested by evolving controllers with the left-rear leg disabled in the simulation. Figure 12 illustrates the symmetry of a resulting champion controller. Surprisingly, it is still very symmetric with the same type of module controlling all four legs. Therefore, it produces a gait similar to a trot (Fig. 13) and sends activation to the disabled leg also. Since the disabled leg does not respond, evolution adapted the gait accordingly to produce a straight walk utilizing only three legs. When this controller is transferred to a similarly disabled physical robot, it produces the same gait on that robot as well, thus demonstrating successful transfer.

5 Discussion and future work

In the above experiments, ENSO evolved controllers that produced the same gaits in the physical robot as they did in simulation. Moreover, these gaits were robust to uncertainties that commonly occur in the real world such as changes in ground friction between different surfaces, the maximum angular velocity that the motors can produce, and the initial positions of legs from which the robot starts walking. The transfer was successful because of two factors: (1) it was possible to simulate the physical



(a) Simulated robot



(b) Real robot

Fig. 13 A gait evolved in simulation with the left-rear leg disabled and transferred to the similarly disabled real robot. The *plots* show the four leg angles of the robot in the first 12 s. They were produced by the controller with the phenotype graph illustrated in Fig. 12. The disabled leg produces a flat line, while the other three legs maintain synchronous and phase-related oscillations resembling a trot gait. However, the synchronous lines for left-front and right-rear legs split slightly from each other between 0° and 180° and the line for the right-front leg curves a little around 0°, indicating adaptation of the gait to produce a straight walk with only three legs. This is a gait that transfers well to the physical disabled robot

characteristics of the robot with sufficient accuracy, and (2) ENSO was able to evolve robust controllers that produce stable gaits.

It is often difficult to transfer controllers evolved in simulation to the real world because it is difficult to simulate the physical robot and its environment accurately [13, 22]. However, this requirement is less critical for the controllers that ENSO evolves because they are coupled cell systems [9, 33, 35, 36]. Such systems are robust to small perturbations and can thus compensate for small deviations from ideal behavior. For example, instead of detailed mesh models of the robot morphology, a crude approximation of its weight distribution was sufficient. Moreover, it turned out sufficient to model the idiosyncratic properties of the motors and uncertainties in their behavior using interpolation and noise.

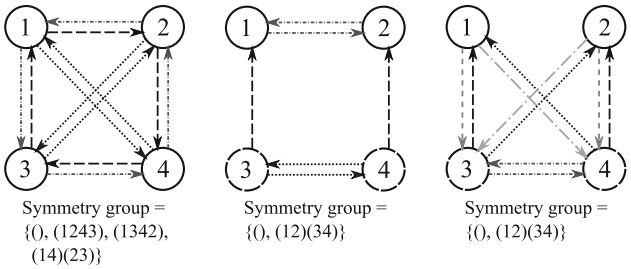


Fig. 14 Variations in phenotype graphs of champion controllers evolved by ENSO. These controllers were evolved using different random number seeds for the experiment discussed in Sect. 4.2. They have different structures either with identical modules (*left graph* and Fig. 7) or different modules for the front and rear legs (*middle* and *right graphs*). However, in all cases, the resulting symmetries were optimized by ENSO to produce effective trot gaits

Modeling the controllers as coupled cell systems also makes it possible to describe the gaits they produce in terms of their symmetries [9]. Based on this observation, ENSO discovers effective gaits by evolving the appropriate symmetry groups for the coupled cell systems. ENSO can discover many such roughly equivalent solutions when initialized with different random number seeds. For example, nine out of ten trials produced unique phenotype graph representations of controllers for the experiment discussed in Sect. 4.2. An arbitrary sample of three such controllers is illustrated in Fig. 14. Although their structures are different, their symmetries were optimized by ENSO to produce similar trot gaits, resulting in roughly the same fitness. Moreover, the controller depicted in the left graph has identical modules, similar to the controller in Fig. 7. In contrast, the controllers depicted in the middle and the right graphs have different modules for the front and rear legs, making it possible to tune the generated gaits if necessary by evolving their behaviors independently. Since different modules can represent different control functions, such graphs are more likely to evolve in more complex platforms that require specialized leg behaviors.

Evolving controllers in simulation and then transferring them to the real robot in this manner is an effective alternative to designing controllers by hand. Hand-design is difficult because it requires anticipating all possible operating conditions. Moreover, this laborious process has to be repeated whenever the configuration of the robot changes and it may even be impossible in some cases. In contrast, evolution can design a new controller automatically for the new configuration in simulation. For example, if a leg actuator fails on a quadruped robot during a remote mission, then it must continue the mission with minimum performance degradation by utilizing only the remaining three legs. ENSO evolved a straight and effective gait for such a three-legged configuration by disabling a leg in simulation. The resulting controller produced the same gait

when transferred to the physical robot, suggesting that the approach presented in this paper can be utilized to design fault-tolerant robots for real-world applications.

The physical quadruped robot used in the experiments can be extended to a hexapod easily by adding two more legs to the attachments in the middle. This extension should be able to demonstrate that the ENSO approach scales up to a physical robot with more legs. ENSO can also be tested on robots with more complex legs such as those with additional joints. Physically realistic simulations have shown that ENSO can evolve effective gaits, e.g. for a quadruped robot with knee joints [36]. The resulting well-coordinated movements that ENSO evolved suggest that it can be extended to evolve controllers for other multi-segmented robot morphologies as well, such as articulated arms and for serpentine robots. For these one-dimensional robots, the search space of symmetries can potentially be reduced by initializing ENSO with controller structures that represent their particular morphologies as closely as possible.

Another interesting extension is to attach Dynamixel AX-S1 sensors to the physical robot to measure distance [30]. Readings from these sensors can then be used as additional controller inputs, making it possible to evolve high-level behaviors that respond effectively to obstacles in the environment.

In addition to extending the robot, the ENSO approach itself can be extended in several ways to evolve controllers more effectively. First, the current manual decomposition of the controller into modules can be automated using hierarchical clustering algorithms based on the robot morphology. Second, instead of using a fixed neural network architecture for the modules, the architectures can be evolved using techniques such as NEAT [32]. Third, crossover of genotype trees can be implemented by swapping subtrees of parent trees if those subtrees have the same structure and node colors. Future work will focus on such extensions in order to eventually provide the sophistication necessary for evolving controllers for real-world robots.

6 Conclusion

This paper demonstrated how the ENSO approach can be utilized to design effective controllers for a physical quadruped robot. Controllers were evolved for an approximate model of the robot in a physical simulation using ENSO and the resulting controllers were then transferred to the physical robot. ENSO makes such transfer tractable by evolving symmetric neural network controllers. Since these controllers are actually coupled-cell systems, they produce stable gaits that are robust to inaccuracies in the simulation and uncertainties in the real world. ENSO evolved effective such controllers both for a fully functional version of the robot and for a version with a disabled leg. Moreover, the results of generalization experiments suggest that these controllers would be robust to common real-world challenges such as variations in battery performance and obstacles.

Acknowledgments Thanks to Ricardo Garcia, Franz Nigl, and Michael Tolley for helping to build the physical robot. This research was supported in part by NSF under grants IIS-0915038, IIS-0757479, and EIA-0303609, and the Texas Higher Education Coordinating Board under grant 003658-0036-2007.

References

1. Beer RD, Chiel HJ, Sterling LS (1989) Heterogeneous neural networks for adaptive behavior in dynamic environments. Adv Neural Inf Proc Syst 1:577–585
2. Beer RD, Gallagher JC (1992) Evolving dynamical neural networks for adaptive behavior. Adapt Behav 1(1):91–122
3. BigDog (2010) http://www.bostondynamics.com/robot_bigdog.html
4. Billard A, Ijspeert AJ (2000) Biologically inspired neural controllers for motor control in a quadruped robot. In: Proceedings of IJCNN, pp 637–641
5. Bongard JC, Lipson H (2004) Once more unto the breach: co-evolving a robot and its simulator. In: Proceedings of ALIFE9, pp 57–62
6. Brooks RA (1989) A robot that walks; emergent behaviors from a carefully evolved network. Technical report AIM-1091, Massachusetts Institute of Technology
7. Brooks RA (1992) Artificial life and real robots. In Proceedings of the First European Conference on artificial life, pp 3–10
8. Clark JE (2004) Design, simulation, and stability of a hexapedal running robot. PhD thesis, Department of Mechanical Engineering, Stanford University
9. Collins JJ, Stewart IN (1993) Coupled nonlinear oscillators and the symmetries of animal gaits. J Nonlinear Sci 3(1):349–392
10. Ferrell C (1994) Failure recognition and fault tolerance of an autonomous robot. Adapt Behav 2(4):375–398
11. Floreano D, Mondada F (1998) Evolutionary neurocontrollers for autonomous mobile robots. Neural Netw 11:1461–1478
12. Gao L, Liu S, Dougal RA (2002) Dynamic lithium-ion battery model for system simulation. IEEE Trans Comp Pack Technol 25(3):495–505
13. Gomez F, Miikkulainen R (2004) Transfer of neuroevolved controllers in unstable domains. In: Proceedings of GECCO
14. Gruau F (1994) Automatic definition of modular neural networks. Adapt Behav 3(2):151–183
15. Holmes P, Full RJ, Koditschek D, Guckenheimer J (2006) The dynamics of legged locomotion: models, analyses, and challenges. SIAM Rev 48(2):207–304
16. Hornby GS, Takamura S, Yokono J, Hanagata O, Yamamoto T, Fujita M (2000) Evolving robust gaits with AIBO. In: Proceedings of ICRA, vol 3, pp 3040–3045
17. Jakobi N (1998) Minimal simulations for evolutionary robotics. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex
18. Jakobi N, Husbands P, Harvey I (1995) Noise and the reality gap: The use of simulation in evolutionary robotics. In Moran F, Moreno A, Merelo J, Chacon P (eds), Advances in artificial life, vol 929. Lecture notes in computer science, pp 704–720
19. Koditschek DE, Full RJ, Buehler M (2004) Mechanical aspects of legged locomotion control. Arthropod Struct Dev 33(3):251–272

20. Kohl N, Stone P (2004) Policy gradient reinforcement learning for fast quadrupedal locomotion. In: Proceedings of ICRA, vol 3, pp 2619–2624
21. Koos S, Mouret J-B, Doncieux S (2010) Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In: Proceedings of GECCO, pp 119–126
22. Lipson H, Bongard J, Zykov V, Malone E (2006) Evolutionary robotics for legged machines: from simulation to physical reality. In: Proceedings of the 9th international conference on intelligent autonomous systems, pp 11–18
23. Mataric M, Cliff D (1996) Challenges in evolving controllers for physical robots. Robot Auton Syst 19(1):67–83
24. Miglino O, Lund HH, Nolfi S (1995) Evolving mobile robots in simulated and real environments. Artificial Life 2:417–434
25. Nolfi S, Floreano D, Miglino O, Mondada F (1994) How to evolve autonomous robots: different approaches in evolutionary robotics. In: Proceedings of artificial life, vol IV, pp 190–197
26. Objet Eden 260V (2010) http://www.objet.com/3D-Printer/Eden260V/
27. ODE: Open dynamics engine (2007) http://www.ode.org/
28. OPAL: Open physics abstraction layer (2007) http://opal.sourceforge.net/
29. Raibert MH (1986) Legged robots. Commun ACM 29(6): 499–514
30. Robotis (2010) http://www.robotis.com/
31. SolidWorks (2010) http://www.solidworks.com/
32. Stanley KO, Miikkulainen R (2004) Competitive coevolution through evolutionary complexification. J Artif Intell Res 21: 63–100
33. Valsalam VK (2010) Utilizing symmetry in evolutionary design. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, technical report AI-10-04
34. Valsalam VK, Miikkulainen R (2008) Modular neuroevolution for multilegged locomotion. In: Proceedings of GECCO, pp 265–272
35. Valsalam VK, Miikkulainen R (2009) Evolving symmetric and modular neural networks for distributed control. In: Proceedings of GECCO, pp 731–738
36. Valsalam VK, Miikkulainen R (2011) Evolving symmetry for modular system design. IEEE Trans Evol Comput 15(3):368–386
37. Watson RA, Ficici SG, Pollack JB (2002) Embodied evolution: distributing an evolutionary algorithm in a population of robots. Robot Auton Syst 39(1):1–18
38. Zagal JC, Ruiz-Del-Solar J (2007) Combining simulation and reality in evolutionary robotics. J Intell Robot Syst 50(1):19–39
39. Zhang SS, Xu K, Jow TR (2003) The low temperature performance of li-ion batteries. J Power Sourc 115(1):137–140
40. Zykov V, Bongard J, Lipson H (2004) Evolving dynamic gaits on a physical robot. In: Proceedings of GECCO, late-breaking papers