Evolving Robot Gaits in Hardware: the HyperNEAT Generative Encoding Vs. Parameter Optimization

Jason Yosinski[1], Jeff Clune[1], Diana Hidalgo[1], Sarah Nguyen[1], Juan Cristobal Zagal[2], and Hod Lipson[1]

[1] Cornell University, 239 Upson Hall, Ithaca, NY 14853, USA
[2] University of Chile, Beauchef 850, Santiago 8370448, Chile
yosinski@cs.cornell.edu

Abstract

Creating gaits for legged robots is an important task to enable robots to access rugged terrain, yet designing such gaits by hand is a challenging and time-consuming process. In this paper we investigate various algorithms for automating the creation of quadruped gaits. Because many robots do not have accurate simulators, we test gait-learning algorithms entirely on a physical robot. We compare the performance of two classes of gait-learning algorithms: locally searching parameterized motion models and evolving artificial neural networks with the HyperNEAT generative encoding. Specifically, we test six different parameterized learning strategies: uniform and Gaussian random hill climbing, policy gradient reinforcement learning, Nelder-Mead simplex, a random baseline, and a new method that builds a model of the fitness landscape with linear regression to guide further exploration. While all parameter search methods outperform a manually-designed gait, only the linear regression and Nelder-Mead simplex strategies outperform a random baseline strategy. Gaits evolved with HyperNEAT perform considerably better than all parameterized local search methods and produce gaits nearly 9 times faster than a hand-designed gait. The best HyperNEAT gaits exhibit complex motion patterns that contain multiple frequencies, yet are regular in that the leg movements are coordinated.

Introduction and Background

Legged robots have the potential to access many types of terrain unsuitable for wheeled robots, but doing so requires the creation of a gait specifying how the robot walks. Such gaits may be designed either manually by an expert or via computer learning algorithms. It is advantageous to automatically learn gaits because doing so can save valuable engineering time and allows gaits to be customized to the idiosyncrasies of different robots. Additionally, learned gaits have outperformed engineered gaits in some cases (Hornby et al., 2005; Valsalam and Miikkulainen, 2008).

In this paper we compare the performance of two different methods of learning gaits: parameterized gaits optimized with six different learning methods, and gaits generated by evolving neural networks with the HyperNEAT generative encoding (Stanley et al., 2009). While some of these methods, such as HyperNEAT, have been tested in simulation (Clune et al., 2009a, 2011), we investigate how they perform when evolving on a physical robot (Figure 1).

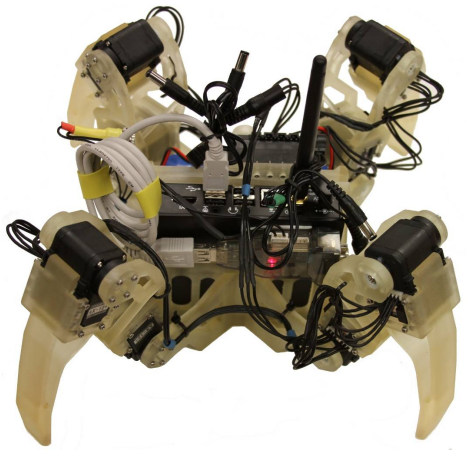

Figure 1: The quadruped robot for which gaits were evolved. The translucent parts were produced by a 3D printer. Videos of the gaits can be viewed at http://bit.ly/ecalgait

Previous work has shown that quadruped gaits perform better when they are *regular* (i.e. when the legs are coordinated) (Clune et al., 2009a, 2011; Valsalam and Miikkulainen, 2008). For example, HyperNEAT produced fast, natural gaits in part because its bias towards regular gaits created coordinated movements that outperformed gaits evolved by an encoding not biased towards regularity (Clune et al., 2009a, 2011). One of the motivations of this paper is to investigate whether any learning method biased towards regularity would perform well at producing quadruped gaits, or whether HyperNEAT's high performance is due to additional factors, such as its abstraction of biological development (described below). We test this hypothesis by comparing HyperNEAT to six local search algorithms with a parametrization biased toward regularity.

An additional motivation is to test whether techniques for evolving gaits in simulation, especially cutting-edge evolu-

tionary algorithms, transfer to reality well. Because HyperNEAT gaits performed well in simulation, it is interesting to test whether HyperNEAT can produce fast gaits for a physical robot, including handling the noisy, unforgiving nature of the real world. Such tests help us better understand the real world implications of results reported only in simulation. It is additionally interesting to test how more traditional gait optimization techniques compete with evolutionary algorithms when evolving in hardware. A final motivation of this research is simply to evolve effective gaits for a physical robot.

Related Work

Various machine learning techniques have proved to be effective at generating gaits for legged robots. Kohl and Stone presented a policy gradient reinforcement learning approach for generating a fast walk on legged robots (Kohl and Stone, 2004), which we implemented for comparison. Others have evolved gaits for legged robots, producing competitive results (Chernova and Veloso, 2005; Hornby et al., 2005; Zykov et al., 2004; Clune et al., 2009a, 2011, 2009b,c; Téllez et al., 2006; Valsalam and Miikkulainen, 2008). In fact, an evolved gait was used in the first commercially-available version of Sony's AIBO robot (Hornby et al., 2005). Except for work with HyperNEAT (Clune et al., 2009a, 2011, 2009b,c), the previous evolutionary approaches have helped evolution exploit the regularity of the problem by manually decomposing the task. Experimenters have to choose which legs should be coordinated, or otherwise facilitate the coordination of motion. Part of the motivation of this paper is to compare the regularities produced by HyperNEAT to those generated by a more systematic exploration of regularities via a parameterized model.

Problem Definition

The gait learning problem aims to find a *gait* that maximizes some performance metric. Mathematically, we define a gait as a function that specifies a vector of commanded motor positions for a robot over time. We can write gaits without feedback — also called open-loop gaits — as

$$\vec{x} = g(t) \tag{1}$$

for commanded position vector $\vec{x}$. The function depends only on time.

It follows that open-loop gaits are deterministic, producing the same command pattern each time they are run. While the commanded positions will be the same from trial to trial, the actual robot motion and measured fitness will vary due to the noisiness of trials in the real world.

For the system evaluated in this paper, we chose to compare open-loop gaits generated by both the parameterized methods and HyperNEAT. An interesting extension would be to allow closed-loop gaits that depend on the measured servo positions, loads, voltage drops, or other quantities.

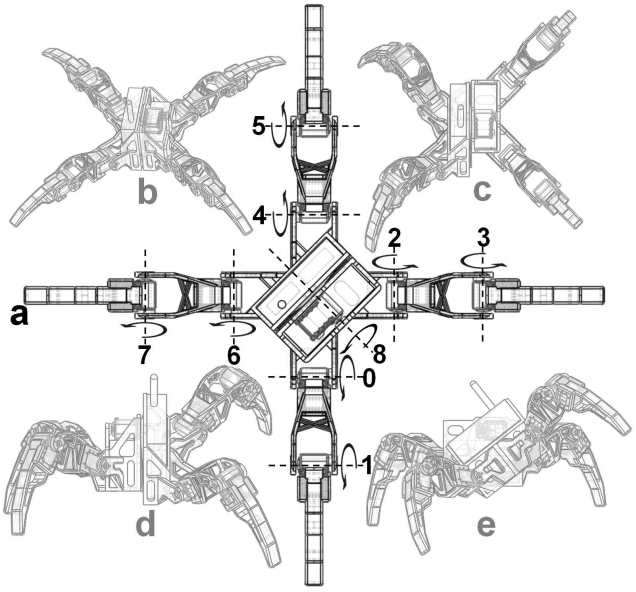

Figure 2: (a) Top-down perspective of the robot with the nine joints and associated servos labeled. (b) The robot in a flat pose with the hip joint centered. (c,d,e) Various views of a pose in which the hip joint is rotated.

The ultimate goal was to design gaits that were as fast as possible. Our performance metric was thus displacement over the evaluation period of 12 seconds. Details of how this displacement was measured are given below.

Experimental Setup

Platform Details

The quadruped robot in this study was assembled from off-the-shelf components and parts printed on the Objet Connex 500 3-D Printing System. It weighs 1.88 kg with the on-board computer and measures approximately 38 centimeters from leg to opposite leg in the crouch position depicted in Figure 1. The robot is actuated by 9 AX-12+ Dynamixel servos: one inner joint and one outer joint servo in each of the four legs, and one servo at the center "hip" joint. This final unique servo allows the two halves of the robot to rotate with respect to each other. Figure 2 shows this unique motion, as well as the positions and numerical designations of all nine servos. Each servo could be commanded to a position in the range [0, 1023], corresponding to a physical range [-120°, +120°]. The computer and servos can be powered by two on-board batteries, but for the tests presented in this paper power was provided by a tethered cable.

All of the computation for gait learning, fitness evaluation, and robot control was performed on the compact, on-board CompuLab Fit-PC2, running Ubuntu Linux 10.10.

Figure 3: A Nintendo Wii remote provided the location of the robot by tracking the infrared LED mounted on the robot's antenna. The position was measured in pixels and transmitted from the Wii remote to the robot via bluetooth.

The slowest portion of code was HyperNEAT, which took less than one second per generation to run (excluding physical evaluations). Thus, we chose not to offload any computation. All gait generation, learning, and fitness evaluation code, except HyperNEAT, was written in Python and is available on our website (http://bit.ly/ecalgait). HyperNEAT is written in C++. We controlled the servos with the Pydynamixel library, sending commanded positions at 40Hz. The robot connected to a wireless network on boot, which enabled us to control it via SSH.

Robot gaits are defined by a Python *gait function* that takes time (starting at 0) as a single input and outputs a list of nine commanded positions (one for each servo). To safeguard against limb collision with the robot body, the control code cropped the commands to a safe range. This range was [-85°, +60°] for the inner leg servos, [-113°, +39°] for the outer leg servos, and [-28°, +28°] for the center hip servo.

Fitness Evaluation Details

To track the position of the robot and thus determine gait fitness, we mounted a Nintendo Wii remote on the ceiling and an infrared LED on top of the robot (Figure 3). The Wii remote contains an IR camera that tracks and reports the position of IR sources. The resolution of the camera was 1024 by 768 pixels with view angles of about 40° by 30°, which produced a resolution of 1.7mm per pixel when mounted at a height of 2.63m. At this height, the viewable window on the floor was approximately 175 x 120 cm.

A separate Python tracking server ran on the robot and interfaced with the Wii remote via bluetooth using the CWiid library. Our fitness-testing code communicated with this server via a socket connection and requested position updates at the beginning and end of each run.

As mentioned earlier, the metric for evaluating gaits was the Euclidian distance the robot moved during a 12-second run on flat terrain. For the manual and parameterized gaits, the fitness was this value. The HyperNEAT gaits stressed the motors more than the other gaits, so to encourage gaits that did not tax the motors we penalized gaits that caused the servos to stop responding. When the servos stopped responding they could, in nearly all cases, be restarted by cycling power, though over the course of this study we did have to replace four servos that were damaged. The penalty was to set the fitness to half of the distance the robot actually traveled. We tested whether the servos were responding after each gait by commanding them to specific positions and checking whether they actually moved to those positions. This test had the additional benefit of rewarding those gaits that did not flip the robot into a position where it could not move its legs, which HyperNEAT also did more than the other learning methods. Because the fitness of HyperNEAT gaits were often halved, in results we compare actual distance traveled in addition to fitness for the best gaits produced by each class of gait-generating algorithms.

Since only a single point on the robot — the IR LED — was measured for the purposes of computing fitness, it was important that the position of the IR LED accurately reflect the position of the robot as a whole. To enforce this constraint, the robot was always measured while in the *ready* position (the position shown in Figure 1). This was done to prevent assigning extra fitness to, for example, gaits that ended with the robot leaning toward the direction of travel (this extra distance would not likely generalize in a longer run, which is why we did not want to reward this behavior).

In order to measure the start and end position in the same pose, and to ensure fair fitness evaluations with as little noise as possible, we linearly interpolated the motion of the robot between the ready position and the commanded gait, $g(t)$. As shown in Figure 4, the instantaneous robot limb configuration during the first and last portions of the evaluation was an interpolation between the initial ready position and $g(t)$; during the rest of the evaluation, the robot followed the commanded gait exactly.

The only human intervention required during most learning trials was to occasionally move the robot back into the viewable area of the Wii remote whenever it left this window. Initially this was a rare occurrence, as the gaits did not typically produce motion as large as the size of the window (roughly 175 x 120 cm). However, as gaits improved, particularly when using HyperNEAT, the robot began to walk out of the measurement area a non-negligible fraction of the time. Whenever it did so, we would discard the trial and

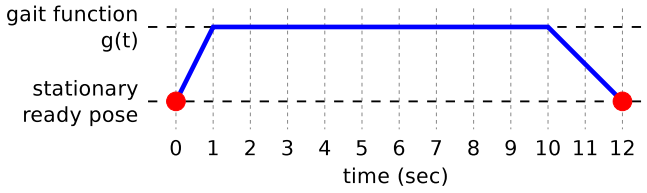

Figure 4: Motion was interpolated linearly between a stationary pose and the commanded gait $g(t)$ for one second at the beginning of each run and two seconds at the end, as shown above. The position of the robot was measured at the beginning and end of each run (red circles) in the ready pose.

repeat it until the gait finished within the window. While this process guaranteed that we always obtained a measurement for a given gait before proceeding, it also biased some measurements downward. Because the performance of the robot on a given gait varied from trial to trial, a successful measurement was more likely to be obtained when the gait happened to perform poorly. This phenomenon was negligible at first, but became more pronounced as gaits began traversing the entire area. HyperNEAT gaits were especially likely to require additional trials, meaning that the reported performance for HyperNEAT is worse than it would have been otherwise. Future studies could employ an array of Wii remotes to increase the size of the measurement arena.

Gait Generation and Learning

We now describe the classes of gait-generating algorithms.

Parameterized Gaits

By a *parameterized gait*, we mean a gait produced by a parameterized function $g(t;\vec{\theta})$. Fixing the parameters $\vec{\theta}$ yields a deterministic motion function over time. We tried several parametrizations on the robot and, upon obtaining reasonable early success, settled on one particular parametrization, which we call *SineModel5*. Its root pattern is a sine wave and it has five parameters (Table 1).

Intuitively, SineModel5 starts with 8 identical sine waves of amplitude α and period τ, multiplies the waves for all outer motors by m_O, multiplies the waves for all front motors by m_F, and multiplies the waves for all right motors by m_R. To obtain the actual motor position commands, these waves are offset by fixed constants ($C_O = 40$ for outer motors, $C_I = 800$ for inner motors, and $C_C = 512$ for the center hip motor) so that the base position (when the sine waves are at 0) is approximately a crouch (the position shown in Figure 1). To keep the size of the model search space as small as possible, we decided to keep the ninth (center) motor at a fixed neutral position. Thus, the commanded position for each motor as a vector function of time is as follows (numbered as in Figure 2):

Parameters in $\vec{\theta}$	Description	Range
α	Amplitude	[0, 400]
τ	Period	[.5, 8]
m_O	Outer-motor multiplier	[-2, 2]
m_F	Front-motor multiplier	[-1, 1]
m_R	Right-motor multiplier	[-1, 1]

Table 1: The *SineModel5* motion model parameters.

$$\vec{g}(t) = \begin{bmatrix} \alpha\cdot\sin(2\pi t/\tau) & \cdot m_F & +C_I \\ \alpha\cdot\sin(2\pi t/\tau)\cdot m_O & \cdot m_F & +C_O \\ \alpha\cdot\sin(2\pi t/\tau) & & +C_I \\ \alpha\cdot\sin(2\pi t/\tau)\cdot m_O & & +C_O \\ \alpha\cdot\sin(2\pi t/\tau) & & \cdot m_R{+}C_I \\ \alpha\cdot\sin(2\pi t/\tau)\cdot m_O & & \cdot m_R{+}C_O \\ \alpha\cdot\sin(2\pi t/\tau) & \cdot m_F & \cdot m_R{+}C_I \\ \alpha\cdot\sin(2\pi t/\tau)\cdot m_O & \cdot m_F & \cdot m_R{+}C_O \\ 0 & & +C_C \end{bmatrix}$$

Learning Methods for Parameterized Gaits

Given the SineModel5 parameterized motion model (see previous section) and the allowable ranges for its five parameters (Table 1), the task is discovering values for the five parameters that result in fast gaits.

If we choose a value for the five dimensional parameter $\vec{\theta}$, then a given physical trial gives us one measurement of the fitness $f(\vec{\theta})$ of that parameter vector. Two things make learning difficult. First, each evaluation of $f(\vec{\theta})$ is expensive, taking 15-20 seconds on average. Second, the fitness returned by such evaluations has proved to be very noisy, with the standard deviation of the noise often being roughly equivalent to the size of the measurement.

We test the ability of different *learning algorithms* to choose the next value of $\vec{\theta}$ to try, given a list of the $\vec{\theta}$ values already evaluated and their fitness measurements $f(\vec{\theta})$.

We evaluated the following six different learning algorithms for the parameterized motion models:

Random: This method randomly generates parameter vectors in the allowable range for every trial. This strategy serves as as baseline for comparison.

Uniform random hill climbing: This method repeatedly starts with the current best gait and then selects the next $\vec{\theta}$ by randomly choosing one parameter to adjust and replacing it with a new value chosen with uniform probability in the allowable range for that parameter. This new point is evaluated, and if it results in a longer distance walked than the previous best gait, it is saved as the new best gait.

Gaussian random hill climbing: This method works similarly to Uniform random hill climbing, except the next $\vec{\theta}$ is generated by adding random Gaussian noise to the current best gait. This results in all parameters being changed at once, but the resulting vector is always fairly close to the

previous best gait. We used independently selected noise in each dimension, scaled such that the standard deviation of the noise was 5% of the range of that dimension.

N-dimensional policy gradient ascent: We implemented Kohl and Stone's (Kohl and Stone, 2004) method for local gradient ascent for gait learning with noisy fitness evaluations. This strategy explicitly estimates the gradient of the objective function. It does this by first generating n parameter vectors near the initial vector by perturbing each dimension of each vector randomly by either $-\epsilon$, 0, or ϵ. Then each vector is run on the robot, and for each dimension we segment the results into three groups: $-\epsilon$, 0, and ϵ. The gradient along this dimension is then estimated as the average score for the ϵ group minus the average score for the $-\epsilon$ group. Finally, the method creates the next $\vec{\theta}$ by changing all parameters by a fixed-size step in the direction of the gradient. For this study we used values of ϵ equal to 5% of the allowable range in each dimension (ranges listed in Table 1), and a step size scaled such that if all dimensions were in the range [0, 1], the norm of the step size would be 0.1.

Nelder-Mead simplex method: The Nelder-Mead simplex method creates an initial simplex with $d + 1$ vertices for a d dimensional parameter space. It then tests the fitness of each vertex and, in general, it reflects the worst point over the simplex's centroid in an attempt to improve it. Several additional rules are used to prevent cycles and local minima; see Singer and Nelder (2009) for more information.

Linear regression: To initialize, this method chooses and evaluates five random parameter vectors. It then fits a linear model from parameter vector to fitness. In a loop, the method chooses and evaluates a new parameter vector generated by taking a fixed-size step in the direction of the gradient for each parameter, and fits a new linear model to all vectors evaluated so far, choosing the model to minimize the sum of squared errors. The step size is the same as in *N-dimensional policy gradient ascent*.

Three runs were performed per learning method. To most directly compare learning methods, we evaluated the different methods by starting each of their three runs, respectively, with the same three randomly-chosen initial parameter vectors ($\vec{\theta}_A$, $\vec{\theta}_B$, and $\vec{\theta}_C$). Runs continued until the performance plateaued, which we defined as when there was no improvement during the last third of a run.

HyperNEAT Gait Generation and Learning

HyperNEAT is an indirect encoding for evolving artificial neural networks (ANNs) that is inspired by the way natural organisms develop (Stanley et al., 2009). It evolves Compositional Pattern Producing Networks (CPPNs) (Stanley, 2007), each of which is a genome that encodes an ANN phenotype (Stanley et al., 2009). Each CPPN is itself a directed graph, where the nodes in the graph are mathematical functions, such as sine or Gaussian. The nature of these functions can facilitate the evolution of properties such as symmetry (e.g. a Gaussian function) and repetition (e.g. a sine function) (Stanley et al., 2009; Stanley, 2007). The signal on each link in the CPPN is multiplied by that link's weight, which can magnify or diminish its effect.

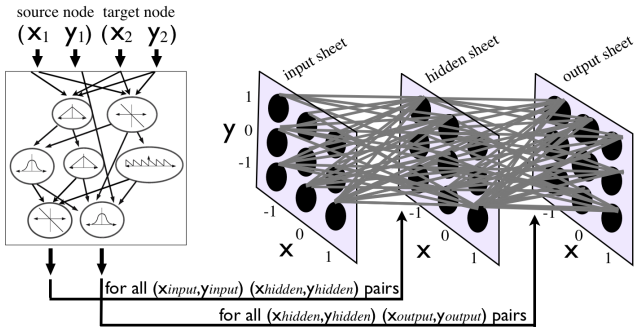

Figure 5: HyperNEAT produces ANNs from CPPNs. ANN weights are specified as a function of the geometric coordinates of each connection's source and target nodes. These coordinates and a constant bias are iteratively passed to the CPPN to determine each connection weight. The CPPN has two output values, which specify the weights for each connection layer as shown. Figure from Clune et al. (2011).

A CPPN is queried once for each link in the ANN phenotype to determine that link's weight (Figure 5). The inputs to the CPPN are the Cartesian coordinates of both the source (e.g. $x = 2$, $y = 4$) and target (e.g. $x = 3$, $y = 5$) nodes of a link and a constant bias value. The CPPN takes these five values as inputs and produces two output values. The first output value determines the weight of the link between the associated input (source) and hidden layer (target) nodes, and the second output value determines the weight of the link between the associated hidden (source) and output (target) layer nodes. All pairwise combinations of source and target nodes are iteratively passed as inputs to a CPPN to determine the weight of each ANN link.

HyperNEAT can exploit the geometry of a problem because the link values between nodes in the ANN phenotype are a function of the geometric positions of those nodes (Stanley et al., 2009; Clune et al., 2009c, 2011). For quadruped locomotion, this property has been shown to help HyperNEAT produce gaits with front-back, left-right, and four-way symmetries (Clune et al., 2009a, 2011).

The evolution of the population of CPPNs occurs according to the principles of the NeuroEvolution of Augmenting Topologies (NEAT) algorithm (Stanley and Miikkulainen, 2002), which was originally designed to evolve ANNs. NEAT can be fruitfully applied to CPPNs because of their structural similarity to ANNs. For example, mutations can add a node, and thus a function, to a CPPN graph, or change its link weights. The NEAT algorithm is unique in three main ways (Stanley and Miikkulainen, 2002). Initially, it starts with small genomes that encode simple networks

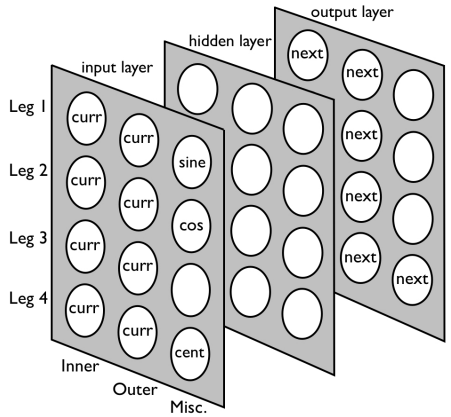

Figure 6: ANN configuration for HyperNEAT runs. The first two columns of each row of the input layer receive information about a single leg (the angles requested in the previous time step for its two joints). The final column provides the previously requested angle of the center joint and, to enable periodic movements, a sine and cosine wave. Evolution determines the function of the hidden-layer nodes. The nodes in the output layer specify new joint angles for each respective joint. The unlabeled nodes in the input and output layers are ignored. Figure adapted from Clune et al. (2011).

and slowly complexifies them via mutations that add nodes and links to the network, enabling the algorithm to evolve the topology of an ANN in addition to its weights. Secondly, NEAT has a fitness-sharing mechanism that preserves diversity in the system and gives time for new innovations to be tuned by evolution before competing them against more adapted rivals. Finally, NEAT tracks historical information to perform intelligent crossover while avoiding the need for expensive topological analysis. A full explanation of NEAT can be found in (Stanley and Miikkulainen, 2002).

The ANN configuration follows previous studies that evolved quadruped gaits with HyperNEAT in simulation (Clune et al., 2011, 2009a), but was adapted to accommodate the physical robot in this paper. Specifically, the ANN has a fixed topology (i.e. the number of nodes does not evolve) that consists of three 3×4 Cartesian grids of nodes forming input, hidden, and output layers (Figure 6). Adjacent layers were allowed to be completely connected, meaning that there could be $(3 \times 4)^2 = 288$ links in each ANN (although evolution can set weights to 0, functionally eliminating the connection). The inputs to the substrate were the angles *requested* in the previous time step for each of the 9 joints of the robot (recall that gaits are open-loop, so actual joint angles are unknown) and a sine and cosine wave (to facilitate the production of periodic behaviors). The sine and cosine waves had a period of about half a second.

The outputs of the substrate at each time step were nine numbers in the range $[-1, 1]$, which were scaled according to the allowable ranges for each of the nine motors and then commanded the positions for each motor. Occasionally HyperNEAT would produce networks that exhibited rapid oscillatory behaviors, switching from extreme negative to extreme positive numbers each time step. This resulted in motor commands to alternate extremes every 25ms (given the command rate of 40Hz), which tended to damage and overheat the motors. To ameliorate this problem, we requested four times as many commanded positions from HyperNEAT ANN's and averaged over four commands at a time to obtain the actual gait $g(t)$. This solution worked well and did not restrict the expressiveness of HyperNEAT.

As with the parameterized methods, three runs of HyperNEAT were performed. Runs lasted 20 generations with a population size of 9 organisms in 3 species, allowing a bare minimum of diversity within and between NEAT species. These numbers were necessarily small given how much time it took to conduct evolution directly on a real robot. The remaining parameters were identical to Clune et al. (2011).

Results and Discussion

Learning Methods for Parameterized Gaits

The results for the parameterized gaits are shown in Figure 7 and Table 2. A total of 1217 hardware fitness evaluations were performed during the learning of parameterized gaits, with the following distribution by learning method: 200 random, 234 uniform, 284 Gaussian, 174 gradient, 172 simplex, 153 linear regression. The number of runs varies because each run plateaued at its own pace. The best overall gait for the parameterized methods was found by linear regression, which also had the highest average performance. The Nelder-Mead simplex also performed quite well on average. The other local search methods did not outperform random search; however, all methods did manage to explore enough of the parameter space to significantly improve on the previous hand-coded gait in at least one of the three runs. No single strategy consistently beat the others: for the first trial Linear Regression produced the fastest gait at 27.58 body lengths/minute, for the second a random gait actually won with 17.26, and for the third trial the Nelder-Mead simplex method attained the fastest gait with 14.83.

One reason the randomly-generated SineModel5 gaits were so effective may have been due to the SineModel5's bias toward regular, symmetric gaits. This may have allowed the random strategy — focusing on exploration — to be competitive with the more directed strategies that exploit information from past evaluations.

HyperNEAT Gaits

The results for the gaits evolved by HyperNEAT are shown in Figure 8 and Table 2. A total of 540 evaluations were performed for HyperNEAT (180 in each of three runs). Overall the HyperNEAT gaits were the fastest by far, beating all the parameterized models when comparing either average

	Average	Std. Dev.
Previous hand-coded gait	5.16	–
Random search	9.40	6.83
Uniform Random Hill Climbing	7.83	4.56
Gaussian Random Hill Climbing	10.03	6.00
Policy Gradient Descent	6.32	7.39
Nelder-Mead simplex	12.32	3.35
Linear Regression	14.01	12.88
Evolved Neural Network (HyperNEAT)	29.26	6.37

Table 2: The average and standard deviation of the best gaits found for each algorithm during each of three runs, in body lengths/minute.

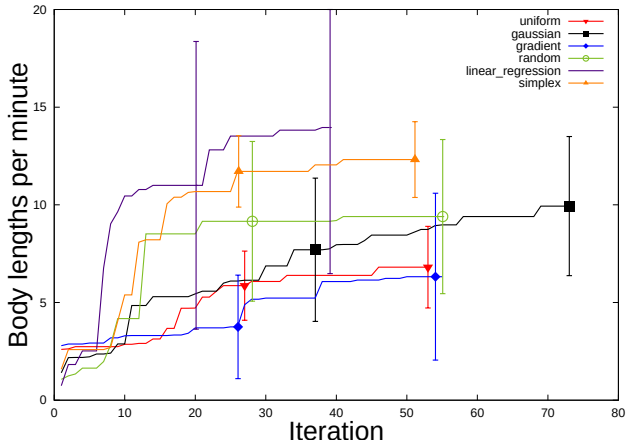

Figure 7: Average results (± SE) for the parameterized learning methods, computed over three separately initialized runs. Linear regression found the fastest overall gait and had the highest average, followed by Nelder-Mead simplex. Other methods did not outperform a random strategy.

or best gaits. We believe that this is because HyperNEAT was allowed to explore a much richer space of motions, but did so while still utilizing symmetries when advantageous. The single best gait found during this study had a speed of 45.72 body lengths/minute, 66% better than the best non-HyperNEAT gait and 8.9 times faster than the hand-coded gait. Figure 9 shows a typical HyperNEAT gait that had high fitness. The pattern of motion is both complex (containing multiple frequencies and repeating patterns across time) and regular, in that patterns of multiple motors are coordinated.

The evaluation of the gaits produced by HyperNEAT was more noisy than for the parameterized gaits, which made learning difficult. For example, we tested an example HyperNEAT generation-champion gait 11 times and found that its mean performance was 26 body lengths/minute (± 13 SD), but it had a max of 38 and a min of 3. Many effective HyperNEAT gaits were not preserved across generations because

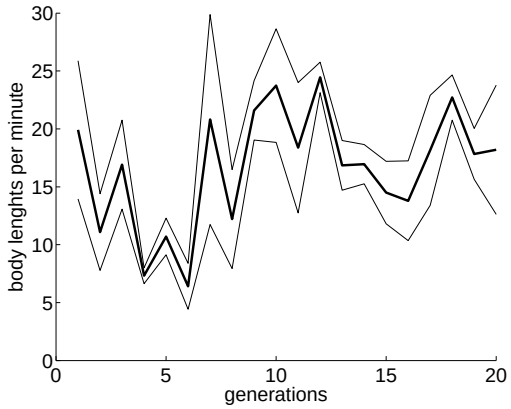

Figure 8: Average fitness (± SE) of the highest performing individual in the population for each generation of HyperNEAT runs. The fitness of many high-performing HyperNEAT gaits were halved if the gait overly stressed the motors (see text), meaning that HyperNEAT's true performance without this penalty would be even higher.

a single poor-performing trial could prevent their selection. The HyperNEAT learning curve would be smoother if the noise in the evaluations could be reduced or more than one evaluation per individual could be afforded.

Conclusion and Future Work

We have presented an array of approaches for optimizing a quadrupedal gaits for speed. We implemented and tested six learning strategies for parameterized gaits and compared them to gaits produced by neural networks evolved with the HyperNEAT generative encoding.

All methods resulted in an improvement over the robot's previous hand-coded gait. Building a model of gait performance with linear regression to predict promising directions for further exploration worked well, producing a gait of 27.5 body lengths/minute. The Nelder-Mead simplex method performed nearly as well, likely due to its robustness to noise. The other parameterized methods did not outperform random search. One reason the randomly-generated SineModel5 gaits performed so well could be because the gait representation was biased towards effective, regular gaits, making the highly exploratory random strategy more effective than more exploitative learning algorithms.

HyperNEAT produced higher-performing gaits than all of the parameterized methods. Its best-performing gait traveled 45.7 body lengths per minute, which is nearly 9 times the speed of the hand-coded gait. This could be because HyperNEAT tends to generate coordinated gaits (Clune et al., 2011, 2009a), allowing it to take advantage of the symmetries of the problem. HyperNEAT can also explore a much larger space of possibilities than the more restrictive 5-dimensional parameterized space. HyperNEAT gaits

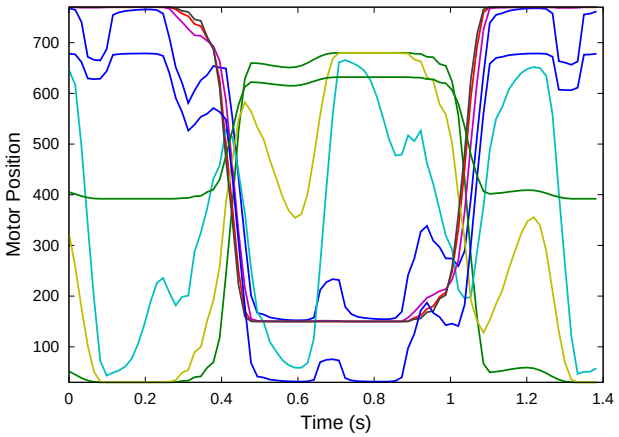

Figure 9: Example of one high-performance gait produced by HyperNEAT showing commands for each of nine motors. Note the complexity of the motion pattern. Such patterns were not possible with the parameterized SineModel5, nor would they likely result from a human designing a different low-dimensional parameterized motion model.

tended to produce more complex sequences of motor commands, with different frequencies and degrees of coordination, whereas the parameterized gaits were restricted to scaling single-frequency sine waves and could only produce certain types of motor regularities.

Because all 1217 trials were done in hardware, it was difficult to gather enough data to properly rank the methods statistically. One direction for future work could be to obtain many more trials. However, a more effective extension might be to combine frequent trials in simulation with infrequent trials in hardware (Bongard et al., 2006). The simulation would produce the necessary volume of trials to allow the learning methods to be effective, and the hardware trials would serve to continuously ground and refine the simulator. One could also guide evolution to the most fertile territory by penalizing gaits that produced large discrepancies between simulation and reality (Koos et al., 2010). Another extension would be to allow gaits that sensed the position of the robot and other variables to enable the robot to adjust to its physical state, instead of providing an open-loop sequence of motor commands. All of these approaches would likely improve the quality of automatically generated gaits for legged robots, which will hasten the day that humanity can benefit from their vast potential.

Acknowledgments

NSF Postdoctoral Research Fellowship in Biology to Jeff Clune (award number DBI-1003220).

References

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

Chernova, S. and Veloso, M. (2005). An evolutionary approach to gait learning for four-legged robots. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2562–2567. IEEE.

Clune, J., Beckmann, B., Ofria, C., and Pennock, R. (2009a). Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2764–2771.

Clune, J., Beckmann, B., Pennock, R., and Ofria, C. (2009b). HybrID: A Hybridization of Indirect and Direct Encodings for Evolutionary Computation. In *Proceedings of the European Conference on Artificial Life*, pages 134–141.

Clune, J., Ofria, C., and Pennock, R. (2009c). The sensitivity of HyperNEAT to different geometric representations of a problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 675–682. ACM.

Clune, J., Stanley, K., Pennock, R., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation. To appear.*

Hornby, G., Takamura, S., Yamamoto, T., and Fujita, M. (2005). Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Transactions on Robotics*, 21(3):402–410.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. *IEEE International Conference on Robotics and Automation*, 3:2619–2624.

Koos, S., Mouret, J., and Doncieux, S. (2010). Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126. ACM.

Singer, S. and Nelder, J. (2009). Nelder-mead algorithm. http://www.scholarpedia.org/article/Nelder-Mead_algorithm.

Stanley, K. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162.

Stanley, K., D'Ambrosio, D., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

Stanley, K. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Téllez, R., Angulo, C., and Pardo, D. (2006). Evolving the walking behaviour of a 12 dof quadruped using a distributed neural architecture. *Biologically Inspired Approaches to Advanced Information Technology*, pages 5–19.

Valsalam, V. and Miikkulainen, R. (2008). Modular neuroevolution for multilegged locomotion. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 265–272. ACM.

Zykov, V., Bongard, J., and Lipson, H. (2004). Evolving dynamic gaits on a physical robot. *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO*, 4.