Exploiting inherent regularity in control of multilegged robot locomotion by evolving neural fields

Benjamin Inden*, Yaochu Jin†, Robert Haschke‡ and Helge Ritter‡
*Research Institute for Cognition and Robotics, Bielefeld University, Germany
Email: binden@cor-lab.uni-bielefeld.de
†Department of Computing, University of Surrey, United Kingdom
‡Neuroinformatics Group, Bielefeld University, Germany

***Abstract*—The control of multilegged robots is challenging because of the large number of sensors and actuators involved. However, the regularity inherent to gait control can be taken into account to design controllers for multilegged robots. In this paper, we show that NEATfields, a method designed for the evolution of large neural networks, can exploit this regularity to evolve significantly better gaits than those evolved by the standard NEAT method. We also show how evolved networks can control a robot with a ball-like morphology to move on a rough terrain. The success in evolving large neural networks suggests that the NEATfields method is a promising tool for studying complex behaviors in robotics and artificial life.**

***Index Terms*—Multilegged Walking, Neuroevolution**

I. Introduction

Neuroevolution [1], i.e. the generation of artificial neural networks by means of evolutionary algorithms, has often been used to automatically create controllers for robots and simulated agents. For example, much research on the evolution of controllers for multilegged robot morphologies has been reported (see [2], [3], [4], [5], [6] and the references therein). In addition, methods to coevolve robot morphology and controllers have resulted in multilegged structures (e.g. [7]). Robots with legs can cross rough terrain, but are quite difficult to control [8], [9].

One problem faced in the evolution of controllers for multilegged robots is that the input and output spaces are quite large. As more neurons and connections are required for control, the performance of evolutionary algorithms typically degrades because the genome that encodes these elements becomes too large to handle with the limited resources that are typically available for artificial evolution. In the work mentioned above, two approaches are typically used to overcome this obstacle. The first approach is to evolve the controller for one leg, and produce replicates for controlling the other legs. Connections between the replicates can also sometimes be evolved. This approach is computationally efficient but requires a priori knowledge and excludes some possible solutions (e.g. those with unequal controllers for different legs) from being found. The second approach is to use an artificial developmental process, also known as *artificial embryogeny*, to grow a large network from a small genome. This method is potentially very powerful, but also presents many challenges [10], [11].

We have recently proposed an alternative to the above approaches that can also generate large neural networks from comparatively small genomes [12], [13]. The basic idea is to evolve networks of neural fields instead of networks of single neurons only. Each element in these fields is a small neural networks evolved using a standard method. As the higher level structure of the network is evolved together with these elements, evolution can discover the regularity inherent in a given task automatically. This approach is more flexible than manually replicating a controller evolved for a single leg, allows the system to generate different controllers for different legs, and allows fine-tuning the individual connection weights.

The NEATfields method is used here for the evolution of controllers for multilegged robots, but the intention is to show that it can more generally be used as a tool for studying cognition. Evolved neural networks have been occasionally used to explore in a relatively bias-free way how certain behaviors can be generated [14], [15]. However, the limitation to small neural networks in most existing research makes it difficult to study complex behaviors, e.g. where visual information processing is involved. The results in this work suggest that NEATfields, along with some other recently proposed methods (e.g. [16]), is a powerful new tool for studying cognition.

II. Methods

A. The NEATfields method

1) General approach: NEATfields is based on the well known NEAT method [17], [18] that simultaneously evolves the topology and the connection weights of neural networks. NEAT starts evolution with the simplest possible network topology and proceeds by complexification: There is a mutation operator that adds neurons (always between two connected neurons), and adjusts the connection weights such that the properties of these connections changes as little as possible. Another operator adds a connection between two previously unconnected neurons. Once a new gene is created by mutation, it receives a globally unique reference number. These numbers

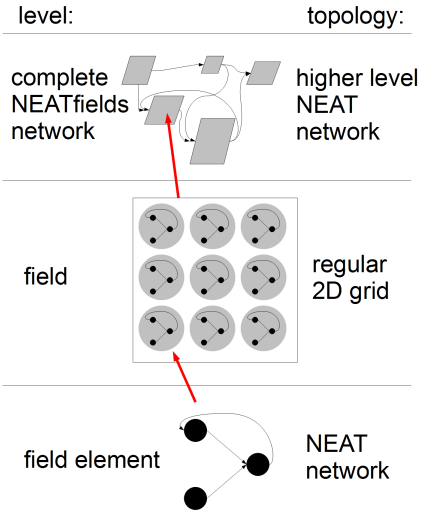

Figure 1. The three-levels architecture in NEATfields.

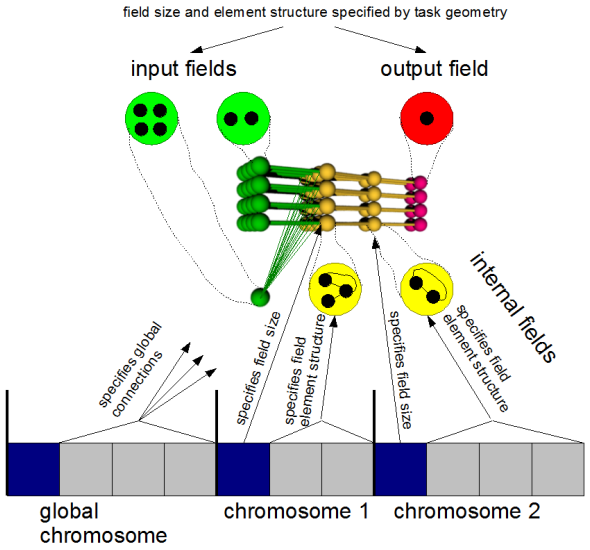

Figure 2. Construction of a NEATfields network from its genome (bottom). The balls in the central network represent field elements. Their contents are shown in the circles above and below. In these circles, black dots represent the individual (input, output or hidden) neurons.

are used to align two genomes during recombination. They are also used to define a distance measure between networks.

The NEATfields method has been designed to extend the NEAT method to the evolution of much larger neural networks. One assumption made by the method is that the input and output spaces of a task can to some degree be decomposed into a number of equal or similar subspaces. Many real-world tasks indeed require one or two dimensional fields of networks that do the same or similar information processing. For example, an eye or a camera provides large amounts of sensory data with a natural two-dimensional topology. Also, robots with actuated limbs often require a number of similar controllers in addition to a coordinating mechanism.

NEATfields networks have a three-level architecture, see figure 1. At the highest level, a network consists of a number of fields that are connected just like individual neurons are in a NEAT network. At the intermediate level, fields are collections of identical (or similar) subnetworks with a two-dimensional topology. At the lowest level, subnetworks, or field elements, are NEAT networks of individual neurons. In accordance with the idea of searching in smaller topologies first, NEATfields usually starts evolution with a single internal field of size 1×1 that has one neuron for each different output occurring in an output field.

2) Neural networks: Similar to many artificial neural networks, the activation level of the neurons in NEATfields is a weighted sum of the outputs of the neurons $j \in J$ to which they are connected, and a sigmoid function is applied on the activation: $o_i(t) = \tanh(\sum_{j \in J} w_{ij} o_j(t-1))$. As in other NEAT implementations, connection weights are constrained to the range $[-3, 3]$. There is no explicit threshold for the neurons. Instead, a constant bias input is available in all networks.

A field element in NEATfields is a recurrent neural network with almost arbitrary topology, which means that no disconnected neurons are allowed and that at most one connection is generated between neurons. A field is a two-dimensional array of field elements. In special cases, the field size along one or both dimensions can be one. A complete NEATfields network is a NEAT-like network where the nodes are fields. It consists of at least one internal field as specified by the genome, and fields for network input and output as specified by the given task. There can be several input and output fields with different dimensions. For example, a bias input can be provided as an input field of size 1×1. Within the NEATfields network, connections can be local (within a field element), lateral (between field elements of the same field), or global (between two fields). It should be noted that connections between field elements or fields are in fact connections between individual neurons in these field elements or fields. How they are established will be described below.

3) Encoding connections and other network elements in the genome: The parameters for an individual field are encoded in the genome in a corresponding chromosome (see figure 2). The first gene in a chromosome specifies the field size in x and y dimensions. The other genes in a chromosome encode the nodes and connections for one field element. In the current implementation, all genes have a length of 160 bit and may contain several numerical parameters and flags as well as unused space for extensions. However, each flag or numerical value within the gene basically has its own specialized mutation operator, so the genes could equally well be implemented as any kind of data structure as long as there are enough bits available for a sufficiently good numerical accuracy. All genes contain a unique reference number that is assigned once the gene is generated by mutation. In addition, connection genes contain a connection weight, a flag indicating whether the connection is active, and the reference numbers of

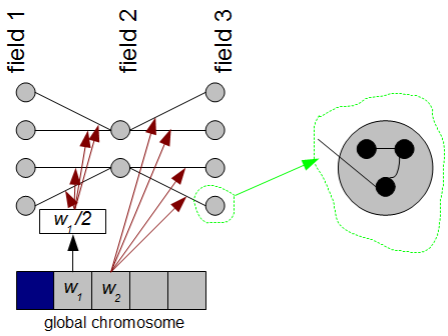

Figure 3. An example of how the global connections between neurons in fields of different sizes (shown here as one dimensional) are created using a deterministic and topology preserving method. The genetically specified weights are automatically scaled if necessary. As shown in detail on the right side, connections go in fact to individual neurons within the field elements as specified by the connection gene.

the source and target neurons (as well as additional data that is explained below). Node genes contain additional values that are not used for the experiments described in this article.

There is a special chromosome that contains genes encoding global connections. Global connections contain reference numbers of a source and a target. These can be reference numbers either of neurons or of inputs and outputs as specified by the task description. They must be in different fields — global connections between two neurons in the same field will never be created by the NEATfields method. Due to the hierarchical architecture of NEATfields, a neuron with a given reference number will be present n times in a field with n field elements. A single global connection gene implicitly specifies connections for all these neurons. If a global connection is between neurons in fields with the same sizes, every field element in the target field will get a connection from the field element in the source field that has the same relative position (in x and y dimension) in its field. Their connection weights are all the same. If field sizes are different in a dimension, then the fields will still be connected using a deterministic and topology preserving method (see figure 3): if the source field is smaller than the target field, each source field neuron projects to several adjacent target field neurons, whereas if the source field is larger than the target field, the target field neurons get input from a number of adjacent source field neurons, while the genetically specified connection weight is divided by that number. This way, field sizes can mutate without changes in the expected input signal strength.

4) Mutation operators for the field element networks: The NEATfields method uses mutation operators that are very similar to those of the original NEAT implementation for evolving the contents of the field elements. The probabilities of these operators are also chosen based on experience with NEAT and the derivative NEON method [19], [18]. The most common operation is to choose a fraction of connection weights and either perturb them using a normal distribution with standard deviation 0.18, or (with a probability of 0.15) set them to a new value. The application probability of this weight changing operator is set to 1.0 minus the probabilities of all structural mutation operators, which amounts to between 0.8815 and 0.949 in the experiments reported here. In general, structural mutations are applied rarely because they will cause the evolutionary process to operate on larger neural networks and search spaces. A structural mutation operator to connect neurons is used with a probability of 0.02, while an operator to insert neurons is used with a probability of 0.001. The latter inserts a new neuron between two connected neurons. The weight of the incoming connection to the new neuron is set to 1.0, while the weight of the outgoing connection keeps the original value. The existing connection is deactivated but retained in the genome where it might be reactivated by further mutations. There are two operators that can achieve this: one toggles the active flag of a connection and the other sets the flag to 1. Both are used with probability 0.01.

5) Evolving network topology on higher levels: For evolving the higher level topology, NEATfields introduces some new operators. At the level of a single field, one operator doubles the field size along one dimension (with a probability of 0.001) and another changes the size of each dimension independently to a random value between its current size and the size of the largest field it is connected to (with a probability of 0.005).

At the level of the complete NEATfields network, there is an operator that inserts global connections (with a probability of 0.01) and an operator that inserts a new field into an existing global connection (with a probability of 0.0005 — higher probabilities will often lead to the evolution of unnecessarily large neural networks). The size of the new field is set randomly to some value between 1 and the larger of the sizes of the two fields between which the new field is inserted. This is done independently for both dimensions. These two operators correspond to operators already used to evolve the topology of the individual field elements. In addition, an existing field can also be duplicated, where all elements of the new field receive new reference numbers. The new field can either be inserted parallel to the old one (in this case, the outgoing connection weights will be halved to prevent disruption of any existing function) or in series with the old one (in this case, every neuron in every field element in the new field receives input from the corresponding neuron in the corresponding field element in the old field, while the output from the new field goes to where the output from the old field went previously). These two operators allow for reuse of previously evolved structure. The serial duplication operator is also applied on input fields, in which case an internal field is created that contains one neuron for every input in the input field. Both mutations occur with a probability of 0.0005.

6) Flow of information within neural fields: NEATfields networks can also have lateral connections between field elements of the same field. These connections enable flow of information within the neural field. Like local connections, lateral connections are between two neurons in the NEAT

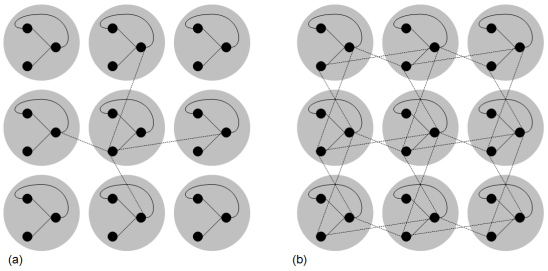

Figure 4. Lateral connections are established between a neuron in a field element and another neuron in each of the up to four neighbor field elements. There are less neighbors if it is at the border of the field. (a) Lateral connections are only shown for the central element as dotted lines here for clarity. (b) All lateral connections constructed from a single gene.

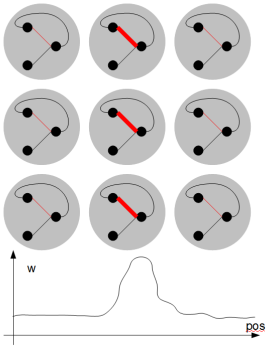

Figure 5. Dehomogenization of neural fields by the focal areas technique. The thickness of connections here symbolizes their weights.

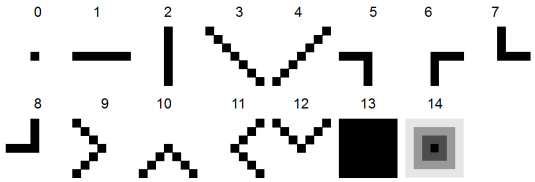

Figure 6. Connection weight patterns for local feature detection. Black denotes a connection weight of 1.0, white a connection weight of -1.0, and the gray scales denote connection weights between 0.0 and 1.0.

network that describes the field elements. However, the connection from the source neuron does not go to a neuron in the same field element, but to the corresponding neurons in the up to four immediate neighbor field elements (see figure 4). The gene encoding a lateral connection specifies source and target neuron reference numbers just as genes encoding local connections do. It is also located in the chromosome that describes its field. However, it has a lateral flag set to 1, and is created by a lateral connect operator (with a probability of 0.02).

7) Dehomogenizing neural fields: By default, corresponding connections in different field elements all have the same strength because they are represented by a single gene. The same is true for the global connections between field elements of two fields. For some tasks, it is useful to have field elements that react slightly differently to the input, which can create what has been called *repetition with variation* [20]. One way to realize this is that connection strengths are larger in a neighborhood of a given position on the field. The NEATfields method can scale connection weights according to $\exp(-\epsilon(\frac{distance}{field\,size})^2)$ (in our implementation, this is done separately for the x and y dimensions), where $\epsilon = 5.0$ is chosen such that connections close to the specified focal position have a large weight and the rest have small weights (see figure 5). The focal position is specified in the following way: There are two eight bit values in the gene encoding a connection, one for each dimension. These are converted to two numbers between -1 and 1. If a number is between -0.15 and 0.15, the connection weights will be homogeneous in the corresponding dimension. This is the default for all connections because they are usually created with these values set to 0.0. If a value is outside this range, on the other hand, then it is mapped linearly to a position between the two field borders. There is a mutation operator that (at a probability of 0.03) sets the values for a single connection gene.

In principle, a field can be completely dehomogenized by many of these *focal area* connections. One even faster way of completely dehomogenizing the weights is to scale them with a random factor. Here, a random factor does not mean that the factors are randomly drawn every time the network is created. Instead, the same random data is retrieved every time the network is created using the innovation number of the gene as a key. This is achieved by passing the innovation number as seed to a common random number generator that then generates the required amount of data. A similar technique was used in [19]. In those experiments that use random dehomogenization, we set a corresponding flag in the genes of 25% of all newly created connection genes. The flag does not mutate in subsequent generations.

8) Building local feature detectors: With ordinary global connections, a field element can only receive input from field elements with corresponding positions in other fields. If the fields are of unequal sizes, a field element can get input from several adjacent field elements in another field as described above, but these inputs all have the same connection weights, so processing is possible in a limited way only. More sophisticated processing is useful for some tasks. For example, if the input field contains camera data, detection of local features such as lines or edges in a particular orientation requires the comparison of adjacent cells of the input. To deal with this requirement, connection genes have an additional number that refers to one of at most 16 prespecified connection weight patterns. These patterns have been designed by hand with the aim of providing useful building blocks for local feature detection (see figure 6). A value from the pattern is multiplied by the original connection weight specified in the

gene to get the weight of the connection from the field element at the corresponding position. The patterns have a size of 7×7. The central element of the pattern corresponds to the weight modification of the connection between the field elements with identical relative positions. The feature detectors do not have to use an entire pattern. Instead, the maximal offsets between the field element positions in the source and target fields can be between 0 and 3 for the x and y dimensions, and are also specified on the genome. There is a mutation operator (used with a probability of 0.02 if local feature detectors are used) that changes the pattern reference and maximal offsets of an already existing connection. Both for existing and newly created connections, choices of pattern use and sizes are made based on probabilities that are externally specified for a given experiment. In the experiments reported below, patterns 1 to 4 from figure 6 are used with equal probabilities by default. The maximal offsets in x and y direction are 0 or 1 with equal probabilities by default. Comparisons to other parameter settings and to a number of alternative approaches have been done previously using other tasks and show that the exact choice of patterns and parameters is not important as long as a reasonably rich set of building blocks is provided [13].

B. Selection methods

Here we use a hybrid selection method that combines fitness based tournament selection and a recent approach known as *novelty search* [21]. We found that this method, and some other hybrid methods, performed quite well on a wide range of different task as compared to standard methods [22]. Half of the population is chosen based on fitness, and the other half is chosen based on novelty. Each half has its own elite of size 10 that is copied to the next generation unchanged.

To calculate novelty, a measure for calculating the distance between two individuals is necessary. Here we use a measure that is tuned to give roughly equal weights to differences in the genotype and the behavior of the individuals. The behavioral distance measure for the tasks presented here is the distance of the final positions of two individuals. Because this distance is often very small in the first generations compared to the expected genetic distance, it is multiplied by 10000 for these particular tasks. The globally unique reference numbers assigned to each gene by NEATfields when it arises by mutation is used to calculate genetic distance as follows:

$$\begin{aligned} d &= c_n \#ref_n + c_r \#ref_c + c_w \sum \Delta w \\ &\quad + c_f \#ref_f + c_s \sum \log(1 + \Delta s_x + \Delta s_y) \end{aligned}$$

where $\#ref_n$ is the number of neuron genes present in only one of these networks, $\#ref_c$ is the number of connection genes present in only one of these networks ($\#ref_n$ and $\#ref_c$ are only counted in fields that are present in both networks, otherwise the excess neuron and connection genes are just ignored), $\#ref_f$ is the number of fields present in only one of these networks, Δw are the connection weight differences (summed over pairs of connection genes that are present in both networks), the Δs are the field size differences in the x and y dimension (summed over pairs of fields present in both networks), and the c variables are weighting constants.

Once all pairwise distances have been calculated, the novelty of an organism can be estimated by taking the mean of its 15 lowest distance values. Only the distances to the 50% less fit individuals are taken into account to calculate novelty to minimize the distortion of novelty based selection by fitness based selection. Selection by novelty, like selection by fitness, is then done with tournament selection. If an individual is above a task specific novelty threshold, it is copied into an archive. In every 10 generations, the number of individuals that have been copied into the archive will be checked. If this number is above 3, the threshold is multiplied by 1.05, whereas if it is below 3, the threshold is multiplied by 0.95. Individuals from both the current population and the archive are taken into account for calculating novelty.

For the experiments reported here, we use an initial genetic novelty threshold of 10.0, and set $c_n = 0.0$, $c_r = 1.0$, $c_w = 1.0$, $c_f = 1.0$, $c_s = 1.0$. A population of 100 individuals is evolved for 1000 generations.

To avoid exhausting the available hardware resources, the fitness and novelty of networks having more than a predefined number of nodes or connections are set to 0.0. The limits obviously depend on the problem, the implementation of the algorithm, and the available hardware. Here we set them to 10000 nodes and 100000 connections.

C. Quadruped walking task

We use a task similar to the one reported in [6], where both the motivation for the problem and previous results are described in more detail. Robots with four legs are simulated using the ODE physics engine for 60 seconds using a time step of 0.01 s. The fitness is calculated based on the total distance travelled by a robot in x and y direction as $(x^2 + y^2)^2$. A trial is aborted if any body segment of the robot other than the four lowest leg segments touches the ground, or if the directions of movement of the joints have been changed more than 960 times altogether. Besides, a trial is aborted if a robot jumps higher than some threshold value because that can only be achieved by exploiting the inaccuracies of the physics simulation. In the latter case, the fitness is set to 0.0.

The robot (see Fig. 7) has a rectangular torso of size $0.15 \times 0.3 \times 0.05$ units. Each of its four legs consists of three cylindrical segments of length 0.075 and radius 0.02. They are connected to the body or their preceding segments by hinge joints. The first joint allows forward and backward movements and is constrained within a maximum of 180°

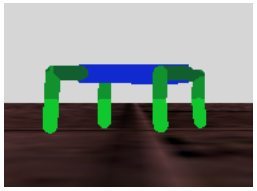

Figure 7. Robot morphology for the legged locomotion task.

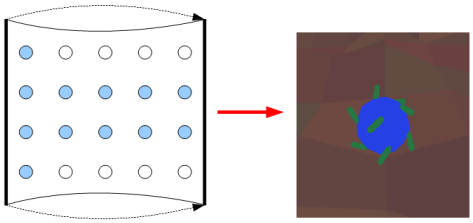

Figure 8. The morphology of legged spheres (right) and how a rectangular input/output field is mapped to the legs. Empty circles denote field elements that are not connected to any actuators in the robot.

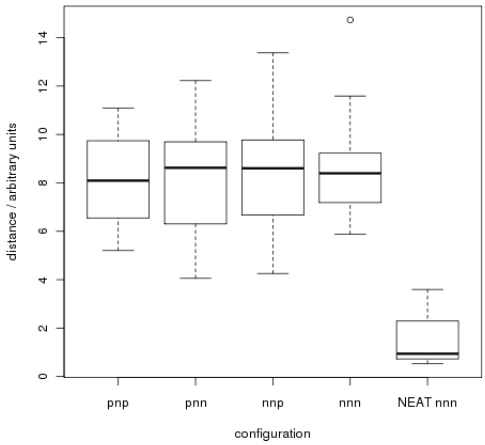

Figure 9. Performance of four legged walking machines. "p" stands for same orientation of a joint on both sides of the body, while "n" stands for different orientations. The letters are given in a proximal-distal order.

rotation. The second joint allows the two lower leg segments to swing towards or away from the body. The third joint, which, like the second, is unconstrained, allows the lowest segment to move forward or backward. The controller gets a 2×2 input field where the element of each leg contains the three joint angles and a touch sensor for the lowest segment. It gets another input field with the yaw, pitch, and roll values of its torso, as well as a bias input. The output field, which is of size 2×2, contains three outputs for the desired joint angles p_i^s for each field element. The simulator takes these to move the joints with a desired speed $v_i = 2.0(p_i - p_i^s)$ and a maximal force of 100 units for every joint i (where p_i is its current position). Unlike in [6], no periodic signal was provided as input because it was found in preliminary experiments that it did not facilitate the evolution of periodic gaits.

In principle, joints on the opposite sides of the robot body can have either the same orientation ("p") or exactly opposite orientations ("n"). For example, an "n" for the first joint would mean that for the same motor command, the legs on one side of the body would swing forward, while those on the other side would swing backward. We used "nnn" as standard configuration, but tested also "pnp", "pnn", and "nnp". Different configurations are expected to bias evolution towards the behavior of different locomotion strategies.

D. Control of legged spheres

In the task decribed now, the robot body is shaped like a ball (with a radius of 0.2) with 12 legs (of length 0.1) surrounding it. The legs consist of a single element that is connected to the ball using a universal joint, so there are 24 degrees of freedom altogether. While there was no exact biological example for this design, tumbleweeds and many other plants produce seeds that are ball-shaped and move with the help of wind [23]. Besides, many sea urchins habitually walk on some of their spines, although they do not roll, but keep their oral side towards the solid surface [24].

A ball-like robot obviously cannot topple over. This is useful for exploration of rough terrains, and a similar design was already suggested for exploration of planetary surfaces [23]. Here we assume that the robot will not be not damaged by rolling down a hill, and that the problem of correctly orienting the body once a location of interest is reached can be solved independently or is not of interest. We focus on the generation of behavior for fast movement. The 12 legs are mapped onto input and output fields of size 5×4 (see Fig. 8). Each field element in this input field has a size of 4 because the legs are controlled by two pairs of antagonistic actuators. The desired velocity of a leg in one dimension is $v(t+1) = \frac{v_{max}}{2}(\frac{o_p(t)-o_n(t)}{2} - \frac{2\alpha(t)}{\pi})$, where $o_p(t)$ and $o_n(t)$ are the antagonistic outputs at time step n, and $\alpha(t)$ is the current joint angle, which is restricted to be within $\pm\frac{\pi}{2}$. So the desired velocity will be within $\pm v_{max}$ in each direction, which is hard-constrained by the physics engine with a defined maximal force.

A terrain was generated using 65×65 points separated by a distance of 0.625 from each other. The heights of the points were derived from $h(x,y) = \frac{r(x,y)+\sqrt{2(x-32)^2(y-32)^2}}{2}$, where the $r(x,y)$ are drawn from a uniform distribution in $[0,1]$, and the other summand grows towards the borders of the terrain. The terrain is enclosed by walls that are high enough to prevent the robot from moving out of the terrain. The fitness assigned to the robot is again the squared distance. The trial is aborted for robots jumping too high or changing joint directions too often as described in the preceeding section. In addition, a check is done every 3 seconds whether the robot's distance from its point of origin is at least $5 \cdot \frac{t}{60\text{s}}$, where t is the current time. If the condition is not fulfilled, the trial is terminated to save simulation time. Results are reported below for NEATfields and and a NEAT configuration where each output is randomly connected to 20% of the input from all legs in the common ancestor.

III. Results

The NEATfields method always finds successful gaits for four legged robots (20 runs were performed for each experiment). On average, they walk a distance of 8.6 length units, whereas robots evolved with NEAT walk only1.5 length units.

In fact, many NEAT runs do not produce any useful walking behavior at all. The difference between NEAT and NEATfields is significant ($p < 10^{-13}$, t-test). Using a very similar setup, it has been reported in [6] that evolved HyperNEAT networks walk to a distance of about 7 on average, while evolved P-NEAT (NEAT with a fixed feedforward architecture) networks walk to a distance of about 3 on average (because both HyperNEAT and P-NEAT are configured as feedforward networks there, the authors provide an additional sine wave signal as input to facilitate the evolution of gaits, whereas we rely on the evolution of recurrent connections or environmental feedback to produce oscillations). Thus NEATfields, like HyperNEAT, can exploit the regularity inherent in the task, thereby gaining an advantage over neuroevolution methods like NEAT that use a direct encoding of connections in the genome. The gaits produced by NEATfields networks are in general very regular (see Fig. 10 (a)). The best evolved NEAT networks also have rather regular gaits, but much slower and less synchronized ones (see Fig. 10 (b)). This is understandable given that improvments must occur independently in all four legs. Other NEAT walkers use very different control strategies for different legs (see Fig. 10 (c)).

As for the other joint configurations, the "pnp" configuration walks 8.2 length units on average, the "pnn" configuration 8.4 length units, and the "nnp" configuration 8.8 length units (see also Fig. 9). Different directions of locomotion evolve in different runs using the same configuration.

An evolved controller for a quadruped is shown in Fig. 11. It can be observed that (a) the network captures the regularity inherent in the task by having the same topology in the internal field elements for the four legs, (b) it uses recurrent connections for control, (c) the four controller instances in the internal field have evolved connections between them, (d) the connections to the legs of the left and the right side of the body have evolved to be slightly different (there is an additional connection to the output field elements for the left side each).

We also found (results not shown) that the task does not become more difficult if we insert a third pair of legs between the other pairs. So hexapod control can also be achieved easily by NEATfields.

Finally, legged spheres travel an average distance of 11.5 with NEATfields, but 1.9 with NEAT ($p < 10^{-14}$, t-test, see also Fig. 12). They travel by rolling, not by just walking on the same few legs all the time.

IV. Discussion

We show that evolution using the NEATfields method can automatically exploit the regularity inherent in the considered tasks. Control stategies for multilegged robots are considered here because they are not only technically interesting, but may be used to explore the generation of locomotion behavior in arthropods [25] and other animals. More generally, the evolution of large networks consisting of neural fields opens up new opportunities for studying complex behaviors in evolutionary robotics. For example, NEATfields networks are well suited

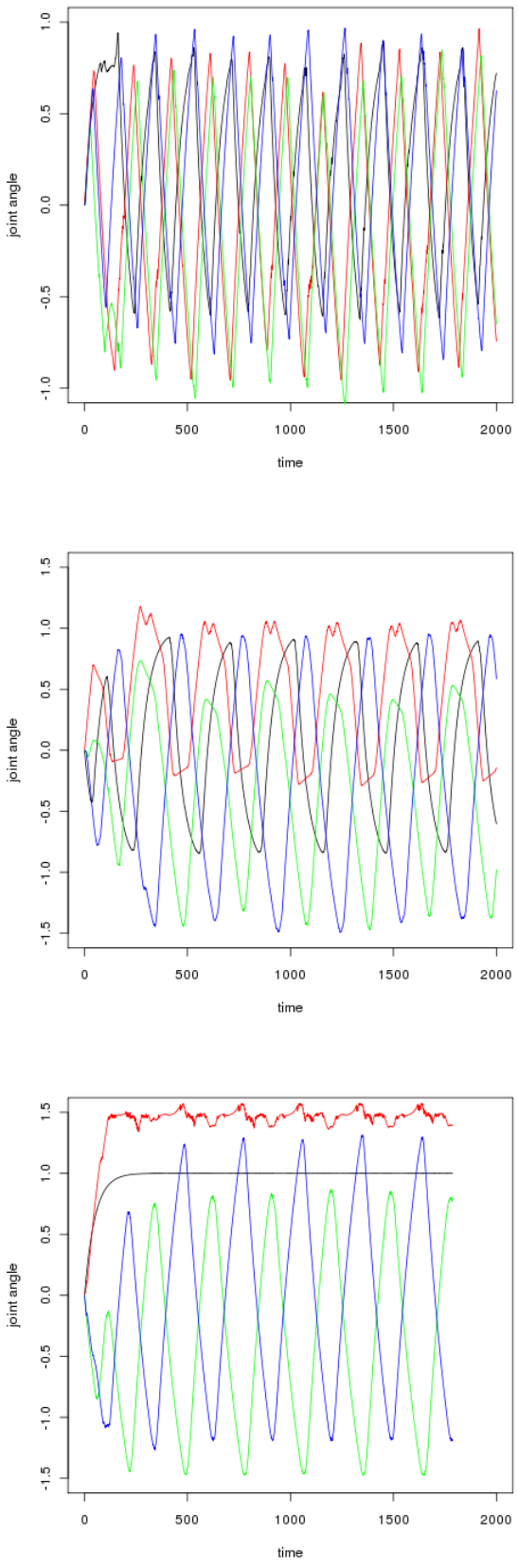

Figure 10. (a) Hip joint angles over time for an evolved NEATfields walker. (b, c) Hip joint angles over time for two evolved NEAT walkers.

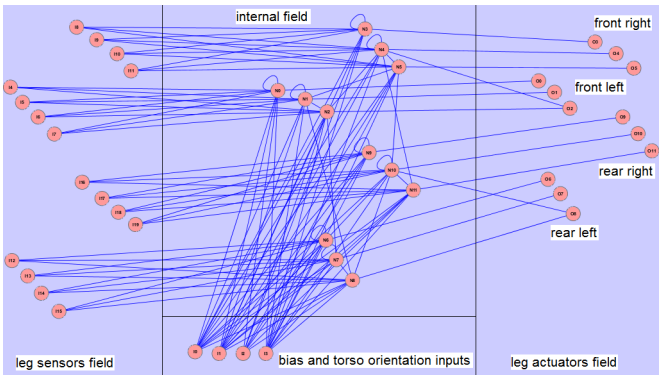

Figure 11. An evolved controller for walking on four legs.

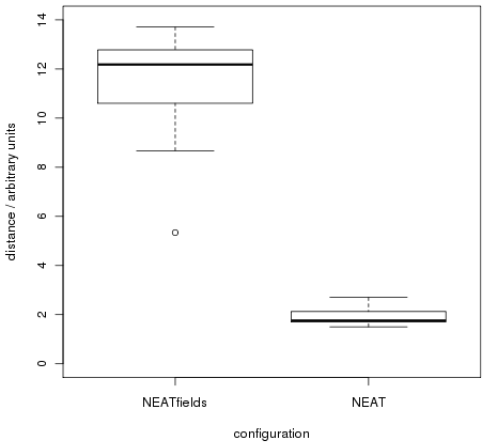

Figure 12. Performance of NEATfields and NEAT on the legged spheres task.

for processing camera data, as shown in our previous work on evolving networks for visual pattern recognition [12], [13]. In addition, the NEATfields method can be used for the evolution of multisensory integration [26]. While it is a complex method with many free parameters, it is also quite robust to changes in many of these parameters [13]. Besides, almost all parameters can remain unchanged for all kinds of different tasks [13], so there is no need for extensive parameter tuning. Further experiments show that the evolution of walking behaviors is also possible with other selection methods besides the one used here [22].

Acknowledgments

Benjamin Inden gratefully acknowledges the financial support from Honda Research Institute Europe. We would like to thank Jeff Clune for providing us with his simulation code as a reference.

References

[1] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, pp. 47–62, 2008.

[2] R. D. Beer and J. C. Gallagher, "Evolving dynamical neural networks for adaptive behavior," *Adaptive Behavior*, vol. 1, pp. 91–122, 1992.

[3] J. Kodjabachian and J.-A. Meyer, "Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle-avoidance in artificial insects," *IEEE Transactions on Neural Networks*, vol. 9, pp. 796–812, 1998.

[4] A. von Twickel and F. Pasemann, "Reflex-oscillations in evolved single leg neurocontrollers for walking machines," *Natural Computing*, vol. 6, pp. 311–337, 2007.

[5] V. K. Valsalam and R. Miikkulainen, "Modular neuroevolution for multilegged locomotion," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2008.

[6] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock, "Evolving coordinated quadruped gaits with the hyperneat generative encoding," in *Proceedings of the IEEE Congress on Evolutionary Computing*, 2009.

[7] T. Miconi, *The Road to Everywhere: Evolution, Complexity and Progress in Natural and Artificial Systems*. PhD thesis, School of Computer Science, University of Birmingham, 2007.

[8] C. Ridderström, *Legged locomotion: Balance, control and tools — from equation to action*. PhD thesis, Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden, 2003.

[9] A. Abbott, "Working out the bugs," *Nature*, vol. 445, pp. 250–253, 2007.

[10] K. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artificial Life*, vol. 9, pp. 93–130, 2003.

[11] S. Harding and W. Banzhaf, *Organic Computing*, ch. Artificial Development. Springer-Verlag, 2008.

[12] B. Inden, Y. Jin, R. Haschke, and H. Ritter, "Neatfields: Evolution of neural fields," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, 2010.

[13] B. Inden, Y. Jin, R. Haschke, and H. Ritter, "Evolving neural fields for problems with large input and output spaces," *submitted*, 2011.

[14] E. Ruppin, "Evolutionary autonomous agents: A neuroscience perspective," *Nature Reviews Neuroscience*, vol. 3, pp. 132–141, 2002.

[15] I. Harvey, E. D. Paolo, R. Wood, and M. Quinn, "Evolutionary robotics: A new scientific tool for studying cognition," *Artificial Life*, vol. 11, pp. 79–98, 2005.

[16] J. Gauci and K. Stanley, "Generating large-scale neural networks through discovering geometric regularities," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2007.

[17] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002.

[18] K. Stanley, *Efficient Evolution of Neural Networks through Complexification*. PhD thesis, Report AI-TR-04-314, University of Texas at Austin, 2004.

[19] B. Inden, "Neuroevolution and complexifying genetic architectures for memory and control tasks," *Theory in Biosciences*, vol. 127, pp. 187–194, 2008.

[20] K. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines*, pp. 131–162, 2007.

[21] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *Proceedings of the Eleventh International Conference on Artificial Life*, 2008.

[22] B. Inden, Y. Jin, R. Haschke, and H. Ritter, "An examination of different fitness and novelty based selection methods for the evolution of neural networks," *submitted*, 2011.

[23] Y. Bar-Cohen and C. Breazeal, "Biologically inspired intelligent robots," in *Proceedings of the SPIE Smart Structures Conference*, 2003.

[24] B. Grzimek and G. M. Narita, eds., *Grzimek's animal life encyclopedia vol. 3: Mollusks and echinoderms*. van Nostrand, 1974.

[25] F. Delcomyn, "Insect walking and robotics," *Annual Review of Entomology*, vol. 49, pp. 51–70, 2004.

[26] B. Inden, Y. Jin, R. Haschke, and H. Ritter, "Evolution of multisensory integration in large neural fields," in *Tenth International Conference on Artificial Evolution*, 2011.