Application of evolved locomotion controllers to a hexapod robot

John C. Gallagher a,*, Randall D. Beer a,b,1, Kenneth S. Espenschied c, Roger D. Quinn c

a Department of Computer Engineering and Science, Case Western Reserve University Cleveland, OH 44106, USA
b Department of Biology, Case Western Reserve University, Cleveland, OH 44106, USA
c Department of Mechanical and Aerospace Engineering, Case Western Reserve University, Cleveland, OH 44106, USA

Abstract

In previous work, we demonstrated that genetic algorithms could be used to evolve dynamical neural networks for controlling the locomotion of a simulated hexapod agent. We also demonstrated that these evolved controllers were robust to loss of sensory feedback and other peripheral variations. In this paper, we show that these locomotion controllers, evolved in simulation, are capable of directing the walking of a real six-legged robot, and that many of the desirable properties observed in simulation carry over directly to the real world. In addition, we demonstrate that these controllers are amenable to hardware implementation and can thus be easily embodied within the robot.

1. Introduction

Designing controllers for autonomous agents can be a very difficult task. Because of this, a number of researchers have suggested using evolutionary techniques to automatically construct controllers based on descriptions of the desired behavior and simulations of the agent [3,6]. When using an evolutionary technique to develop a controller, it becomes almost necessary to operate on a simulation of the device. Thousands of trials may be needed during a given run, and it may simply be impractical to accomplish this on real hardware. Indeed, the “evolutionary failures” that will be seen along the way might be so inappropriate that they could damage the device if one were to allow them control of the unit. Certainly, as pointed out by a number of researchers [4,12], a large dependence on simulation can lead to difficulty. One could argue that evolved controllers could fail to behave appropriately in the presence of interactions not modeled during evolution.

In previous work, we used genetic algorithms (GAs) to develop dynamical neural network controllers to direct the walking behavior of a simulated hexapod agent [3,8]. We have also constructed a six-legged robot [11] to demonstrate the properties of several non-evolved, hand wired, controllers [2,5,7]. In this paper, we will combine the evolved controllers and the robot to address the following two sets of issues:

(1) When replacing the simulated body with a real robot, we may be adding interactions that could disrupt the controller’s ability to produce appropriate behavior. We will refer to this as the body instantiation problem. We will demonstrate that one can indeed develop controllers for non-trivial mechanical devices in simulation and expect them to work in the real world. In fact, with appropriately chosen agent models and controller technology, predictions about controller operation made in simulation will hold in the real world.

*Corresponding author. Tel.: (216) 368-2816; fax: (216) 368-2801; e-mail: johng@alpha.ces.cwru.edu.
1E-mail: beer@alpha.ces.cwru.edu.
Depending on the controller technology used, it may be difficult or impossible to implement it in a way that would be easily carried inside the body itself. We will refer to this as the controller instantiation problem. We will demonstrate that evolved dynamical neural networks are particularly well suited to instantiation as simple electrical circuits by presenting an implementation that uses traditional off the shelf components. We will also briefly examine the potential to construct these circuits using VLSI techniques. We will further show that our evolved controllers are tolerant of the types of fabrication errors we would expect in either implementation method.

In the following sections, we will present brief reviews of previous work in which we constructed the evolved dynamical neural network controllers and the hexapod robot. Following this, we will explain how the controller, which was not evolved specifically for the robot, was interfaced to it. Next, we will address both the body and controller instantiation problems. For the body instantiation problem, we will present simulation results predicting the controller’s ability to deal with real world adversity and use the robot to show that these predictions hold. For the controller instantiation problem, we will sketch an electrical circuit implementation of the controller and present simulation results showing that it can operate in the presence of the kinds of errors that fabrication could introduce into the control circuitry. Finally, we will discuss some implications of the issues raised here, and some possibilities for additional work.

2. Evolved controllers and the robot – A review

2.1. Evolved locomotion controllers

We have successfully used genetic algorithms to develop dynamical controllers for a variety of autonomous agents [3]. In these experiments, we assumed that the dynamics of both the agent’s body and environment were fixed ahead of time. We also assumed that there existed some scalar measure of each agent’s performance. Genetic algorithms were used to maximize agents’ performance measures. This work makes use of controllers we developed to direct the walking of a simulated hexapod agent. In this section, we will briefly review relevant portions of that work.

2.1.1. Simulated agent body model

We used an insect-like agent that is an elaboration of a model used in earlier work (Fig. 1, [1]). Each leg has a foot that can be either up or down. The agent is considered to be statically stable as long as its center of mass remains inside the polygon of support formed by the positions of the supporting feet. As long as the insect is stable, it is considered to be standing and may make forward progress. If stability is lost, the agent falls and its forward velocity is immediately set to zero.

Each leg contains three effectors. The first governs the state of the foot, and the other two generate clockwise and counterclockwise torques about the leg’s single joint. The torques about each joint are summed, and depending on the state of each leg’s foot will either translate the body (foot down) or rotate the leg about its joint (foot up). Each leg has a limited range of angular motion. A supporting leg may stretch outside of this range, but provides no translational forces if it does. In addition to the three effectors, each leg also contains a sensor that measures the leg’s angular position in radians.

2.1.2. Dynamical neural networks

We make use of Hopfield continuous model neurons [9]. Each neuron has the activation function:

\[ \tau \frac{dy_i}{dt} = -y_i + \sum w_{ij} \sigma_j(y_j) + I_i(t), \]  
\[ \sigma_j(x) = (1 + e^{(\theta_j-x)})^{-1}, \]  

Where \( y \) can be interpreted as the mean membrane potential of the neuron, Eq. (2) is a sigmoidal
(S-shaped) function which can be interpreted as its short-term average firing frequency, Θ is a bias constant that controls the firing threshold, τ is a time constant that is associated with the passive properties of the cell membrane. \( w_{ji} \) represents the strength of connection from the \( j \)th neuron to the \( i \)th neuron, and \( I_i(t) \) represents an external input into the neuron. Each leg is controlled by a five-neuron fully interconnected network of the above units. Three of these are motor neurons and govern the state of the forward and backward joint torques and the state of the foot. The remaining two are interneurons with no pre-specified role. Each neuron also receives a weighted sensory input from that leg's angle sensor. Controllers for individual legs are connected to corresponding neurons in the controllers for adjacent legs.

2.1.3. Simulation results

We successfully evolved locomotion controllers capable of properly directing the simulated agent's behavior under three different conditions. Sensory feedback was either (1) always available, (2) never available or (3) available only 50% of the time. Removal of sensors from a controller evolved with access to a sensory signal resulted in severe degradation of performance, often terminating in failure. Controllers evolved without access to sensory information would work, as one would expect, without sensory information - but of course they were not capable of taking advantage of sensory information if it were available. Controllers evolved by averaging the agent performance with and without sensors resulted in mixed controllers that were capable of operating both with and without sensors. As one might expect, the walking behaviors produced by mixed controllers is generally better with its sensors than without (Fig. 2). With sensors intact, mixed controllers generally exhibit higher stepping frequency and cleaner phasing than they do without sensors. Even without sensors, however, the performance of these controllers is generally quite good. The mixed controller shown in Fig. 2 is the one that we will focus on in this paper.

2.2. The robot

The hexapod robot (Fig. 3) utilized here [11] was originally designed to test the real-world capabilities of other non-evolved locomotion controllers [2,7,11]. Each leg has two degrees of freedom: rotation and extension. A leg can swing through 45° from vertical forward or backward. The radial motion is accomplished by means of a rack-and-pinion transmission. Both degrees of freedom are driven by 2 W DC motors with associated integral transmissions. Leg angle sensors are implemented using potentiometers mounted in parallel with the motors. Locomotion controllers were simulated on a personal computer and were interfaced to the robot via eight bit D/A and A/D converters.

3. Controller and body interface

There are two important differences between the simulated body and the robot that needed to be addressed before the evolved controller could be successfully interfaced to the robot.

First, without correction, the swing and radial degrees of freedom would be completely independent in the robot. In the simulated body however, stancing legs passively stretch between their joints and their feet as the body translates. This behavior is captured in the robot by adjusting the radial length of the leg using the kinematic transform: \( r = h / \cos \theta \), where \( h \) is the
height the foot should be above the ground, $\theta$ is the leg's joint angle, and $r$ is the distance the radial degree of freedom should be extended. Using this kinematic transformation, the robot's body can be maintained at a near constant height as it walks.

The second interface issue involves translating between the output that the evolved network generates and the control signals that the robot expects. The dynamical neural network controller supplies signals specifying what torques should be applied at each joint. The robot was designed for use with a different controller and its controller interface circuitry expects to be told desired leg position. The robot's controller interface uses analog linear proportional control circuitry to drive the motors with voltage proportional to the difference between an effector's desired position and its actual position. Knowing a leg's current position and the torque we wish to apply at that leg's joint, we can solve for a target leg position that will produce the desired torque. This computation boils down to scaling by a constant, so interfacing the dynamical neural net to the robot was, therefore, just a matter of adjusting proportionality constants to match the dynamical neural net's time scale to that of the robot.

4. Body instantiation problem

Having a physical body may present difficulties that were not modeled in the simulated body with which the controller was evolved. In this section, we will demonstrate that specific predictions made in simulation about a particular controller's operation hold when that controller is embodied in a physical robot. Specifically we will show that both the robot's normal walking behavior, and the effects sensor noise and damage have on that behavior, are consistent with what simulation predicts. First we will discuss what is predicted in simulation, and then show how the robot's behavior fulfills those predictions.

4.1. Simulation results

In the artificial insect, we saw that the controller we are using generated appropriate control signals to drive the agent's locomotion behavior (Fig. 2). We also saw that the simulated insect walked faster and with better phasing of its legs when sensory information was present, but still operated without the benefit of this information.
Fig. 4. The effects of adding Gaussian noise to the sensors. Each point represents the average distance walked in a set time period for increasing amounts of Gaussian noise. Almost no difference in the simulated insect's behavior is observed until the maximum noise grows to eight times the effective range of the sensor.

We could also assume a robot might encounter a partial sensor loss in which some subset of the leg angle sensors fail. Because the neural controller was never exposed to partial failures of this form during evolution, there is no a priori reason to assume that it should be able to deal with these deficits at all. We examined the effects of removing all subsets of sensors of size 1, 2 and 3 from the simulated artificial insect. In all cases, a deterioration of the walking behavior was seen. In fact, the simulated insects often fell down after every step because those legs with sensors were stepping slightly faster than those without. In no case, however, was the controller's ability to provide rhythmic tripod gaits destroyed.

Even if all the leg sensors are intact, we cannot expect them to always provide faithful representations of each leg’s angular position. Either electrical or mechanical noise could taint the angle readings received from the potentiometers. We examined the effects of adding Gaussian noise to all of the angle sensor values. We ran six sets of experiments with Gaussian noise whose variance ranges from $\frac{1}{2}\pi$ to $8\pi$. For each set of experiments, 1000 trials were run and the average distance walked in a set amount of time was recorded (Fig. 4). The simulated insect walks even in the presence of potentially large sensor errors. The individual neurons act like capacitive filters on the sensor input, effectively averaging over spurious noise.

4.2. Robot results

In Section 4.1 we briefly discussed the controller’s operation in the artificial insect and observed that it worked both with and without sensors and that the artificial insect both walked faster and with better phasing when sensory information was present. Indeed, these same observations were made when the controller was interfaced to the physical robot (Fig. 5). With sensory information available, the robot walked with a smoothly flowing tripod gait. Without sensory information, the robot still walked, but with a punctuated tripod gait that had noticeable pauses between steps in which the robot came to a complete halt.
Next, we examined the effects of removing subsets of angle sensors from the robot. We checked the robot's walking after removal of all sensor subsets of one, two, and three. We did not, in any case, observe the robot to fail down. This brings out an important difference between the simulated agent and the real robot that actually works in our favor. The simulated agent falls over as soon as static stability is lost. The real robot, being an object in the real world, needs time to fall. If stability is lost, but regained quickly enough, the robot will not have time to fall. Seemingly, the minor errors in gate phasing introduced by partial sensor loss are not enough to cause the robot to fail catastrophically.

Our simulation results predicted that the controller would be extraordinarily tolerant of sensor noise when installed in the robot. Indeed the potentiometers in the actual robot were much too well behaved for us to observe any noise induced deficit at all.

5. Controller instantiation problem

For robots to achieve complete autonomy, it is necessary for them to actually carry their controllers onboard. Some controller technologies may require a great deal of hardware to implement, making them too heavy or energy inefficient to be practical to be placed onboard an autonomous robot. If we wish to shrink the size of the robots, this problem becomes even more acute. Electrical circuit implementations are relatively small, making them excellent choices for autonomous agent control. The possibility of implementation using VLSI techniques only increases this potential advantage. In this section, we demonstrate that dynamical neural networks of the sort that we have employed here can be implemented electrically using existing techniques and fabrication processes. In addition, we present an error analysis of our evolved controllers which demonstrates that their sensitivity to parameter variations is well within the errors typically encountered in our circuits.

5.1. Simplified neuron circuit model

A simplified electrical model of a Hopfield model (Fig. 6) could be described as having three parts, an adder, that takes the weighted sum of incoming signals, an integrator, that takes the time integral of that weighted sum, and a mapping circuit, that passes the output of the integrator through the neuron's transfer function (in this case, a sigmoid or hyperbolic tangent function). We shall present electrical implementations of each of these components in turn.

5.2. Adder and integrator stages

The adder can be implemented fairly simply with a standard opamp inverting adder followed by an inverting stage. Each of the inputs \( V_0 - V_4 \) are voltage signals that come from other neurons or external inputs and may be positive or negative. The relative weight of each signal may be adjusted by changing its corresponding variable resistor. The inverse of the weighted sum of the input voltages is presented at node \( A_1 \), where it is fed into an inverting amplifier to restore the proper polarity. The polarity corrected weighted sum at \( A_2 \) is fed into an RC network that very roughly implements the integrator. The time constant of the circuit is equal to the RC value of the final RC stage. The RC network could be replaced with an active opamp based integrator if passive integration proves inadequate for a given application (see Fig. 7).

5.3. Transfer function - An opamp based design

The final stage needs to take in a voltage ranging from approximately -15 to 15 v, and return a voltage proportional to the sigmoid of the input voltage. The following circuit implements the hyperbolic tangent

---

Fig. 6. Simplified electrical model of the Hopfield continuous model neuron.
function electrically, and allows the user to adjust a number of important operational parameters.

We generate a current that is proportional to the hyperbolic tangent of the input voltage using a long tail amplifier. This current is passed into a transimpedance amplifier (converts current to voltage). The gain of the transimpedance amplifier ($R_{\text{gain}}$) can be used to adjust the difference in voltage seen between the top and bottom asymptotes of the hyperbolic tangent. Since the transimpedance amplifier returns a polarity inverted signal, we must invert it using another amplifier. The second amplifier stage inverts the signal, and is used to properly zero the output. Adjusting $R_{\text{offset}}$ moves the whole curve up and down the $Y$-axis and can be used to put the hyperbolic tangent’s lower asymptote at zero (see Fig. 8).

If we need the ability to specify a $X$-offset to the hyperbolic tangent, we can place a simple adder before $V_{\text{in}}$ to add the offset in before the function is computed.

5.4. Complete neuron

A complete neuron can be had by feeding the output of an adder/integrator stage into a map stage. The output of the map stage can be passed to one additional opamp inverter so that both positive and negative versions of the neuron’s activation level may be presented to other neurons.

5.5. VLSI implementation

The transfer functions (Fig. 9) of dynamical neural networks can be directly implemented using standard VLSI fabrication techniques using Carver Mead’s follower-integrator circuits [10]. Work is underway to produce a single chip implementation of the controller. Preliminary layouts (not included here due to space constraints) indicate that fewer than 300 transistors and less than $2000 \times 3000\mu$ of chip area would be needed to implement the entire controller described in this paper.

5.6. Controller parameter tolerance study

Naturally, fabrication processes are not perfect. Therefore, it is important to verify that the neural networks we choose to implement are not so sensitive
to parameter variations that they fail due to unavoidable inaccuracies in fabrication. To test this, we used simulation to study the effects of adding Gaussian error to the time constant and resistor values in our controllers. Eight sets of experiments were run to sample the effects of varying levels of confidence in the fabrication process. For each set, a maximum percent fabrication error was selected, and a random function scaled to the maximum percent error for that test set was used to perturb all the parameters for the controller. For each set, 1000 trials were run and the average distance walked in a set time was recorded (Fig. 10).

It is clear that the distance the robot walks gradually decreases as the amount of fabrication error is increased. Qualitatively, interleg coordination slowly degrades with increasing fabrication errors. However, the difference is hardly noticeable until 20% fabrication error is surpassed. Even after that point, however, the controller never fails entirely and still provides good control of stepping.

6. Discussion

In this paper, we have considered some of the issues raised by attempts to apply autonomous agent controllers evolved in simulation to the control of physical robots. Some have argued that such attempts to apply simulation results to the real world are fraught with difficulties [4,12]. In contrast, we found that applying locomotion controllers evolved in simulation to an actual hexapod robot was a fairly straightforward process. Despite the fact that hexapod robot locomotion is a non-trivial dynamic control problem involving an essential feedback loop through the physical world, and despite the fact that such physical characteristics as inertia, noise and delays were not modeled in our earlier simulation, the evolved controllers performed extremely well in the real world. In addition, we have shown that the results of experiments performed in simulation, such as those involving sensor noise or outright sensor loss, can be quite predictive of real world performance. If anything, the predictions made in simulation were actually rather conservative due to the overly difficult model of static stability that we employed during the original evolution of these controllers.

A second issue that we have examined in this paper concerns the physical implementation of dynamical neural networks. In all of our previous robotic work, controllers were simulated on a separate PC that interfaced to the robot through A/D and D/A converters. Using standard circuit construction techniques, we have proposed an implementation of these evolved controllers that would allow a truly autonomous robot. Although our preliminary sketch does not constitute a complete design, it does go some distance toward showing that such an implementation is indeed feasible. Finally, we have demonstrated that the sensitivity of the evolved controllers to variations in their parameters are well within the error tolerances of either a VLSI or a traditional circuit implementation. In conclusion, although the generality of our experience remains to be examined, our results suggest that simulation experiments can in fact be powerful tools in the design of controllers for real-world robots. More specifically, evolving dynamical neural network controllers in simulation appears to be a viable design methodology for physical robots as well.

Acknowledgements

This work was supported by grant N00014-90-J-1547 to RDB from the Office of Naval Research and Grant NGL-50588 from NASA Goddard. Additional support was provided by the Cleveland Advanced Manufacturing Program through the Center for Automation and Intelligent Systems Research and the Howard Hughes Medical Institute.
References


---

Randall D. Beer received his Ph.D. in computer science from Case Western Reserve University in 1989. He is currently an associate professor at CWRU in the department of computer engineering and science and in the department of biology. He is interested in the evolution and analysis of dynamical neural networks for autonomous agents, biologically inspired robotics, neuroethological modeling, and theoretical issues in adaptive behavior.

Kenneth S. Espenschied received his Ph.D. degree in mechanical engineering from Case Western Reserve University in 1994. Dr. Espenschied is currently working as a research and development engineer at Advanced Robotics Technologies in Chagrin Falls, OH.

Roger D. Quinn received his Ph.D. degree in engineering mechanics from Virginia Polytechnic Institute and State University, Blacksburg, in 1985. He is an associate professor on the faculty of the Mechanical and Aerospace Engineering Department at Case Western Reserve University, having joined the department in 1986. He is also a member of the Center for Automation and Intelligent Systems Research at CWRU. His research is devoted to the study of the dynamics and control of multibody mechanical systems such as machines and animals that walk.

John C. Gallagher received his M.S. in computer science from Case Western Reserve University in 1991, and currently finishing his Ph.D. in the same department. He is interested in dynamical systems analysis of continuous valued neural networks and in the study of the neural basis of cognitive behavior.