Neural Networks Applied to Gait Control of Physically Based Simulated Robots

Milton Roberto Heinen and Fernando Santos Osório
Universidade do Vale do Rio dos Sinos (UNISINOS) - Applied Computing
mheinen@turing.unisinos.br, fosorio@unisinos.br

Abstract

This paper describes our experiments with autonomous robots, in which we use neural networks to generate and control stable gaits of simulated legged robots into a physically based simulation environment. In our approach, the gait is accomplished using an Elman network trained using a gradient descend method, more specifically, the RPROP algorithm, a improvement of the traditional Back-propagation. The model validation was performed by several experiments realized with a simulated four legged robot using the ODE physical simulation engine. The results showed that it is possible to generate stable gaits using neural networks in an efficient manner.

1 Introduction

Control of locomotion in legged robots is a challenging multidimensional control problem[6, 2]. It requires the specification and coordination of motions in all robots' legs while considering factors such as stability and surface friction[14]. This is a research area which has obvious ties with the control of animal locomotion, and it is a suitable task to use to explore this issue[23]. It has been a research area for a considerable period of time, from the first truly independent legged robots like the Phony Pony built by Frank and McGhee[17], where each joint was controlled by a simple finite state machine, to the very successful algorithmic control of bipeds and quadrupeds by Raibert[22].

Lewis[16] evolved controllers for a hexapod robot, where the controller was evaluated on a robot which learn to walk inspired on insect-like gaits, after a staged evolution where its behavior was shaped toward the final goal of walking. Bongard[3] evolved the parameters of a dynamic neural network to control various types of simulated robots. Busch[5] used genetic programming to evolve the control parameters of several robot types. Jacob[13], on the other hand, used reinforcement learning to control a simulated tetrapod robot.

In these previous cited research works, when they involved solutions based on neurons, they all used only the topology of a neural network, but not the learning algorithms based on gradient descend, like back-propagation and its improvements. One of the techniques most used to adjust the neural weights are the genetic algorithms (GA)[8, 18], because they do not need local information for the error minimization, nor the gradient calculation. This is very useful in the robot control, because it is very difficult to have some training data for the supervised learning.

In our previous work[11, 12], we used genetic algorithms [8] to automatically generate a finite state machine to control simulated legged robots with four and six legs. Despite of its advantages, genetic algorithms are a machine learning method[19] not much suitable for the control generation of legged robots, because the time spent to evolve the parameters is very long[26].

For this reason, we decided to use a supervised learning method to adjust the neural weights, more specifically, we adopted the RPROP algorithm[24]. This training method is more efficient than the evolutionary methods, and it is more robust faced to non expected situations than hand tuned methods.

2 Mobile robot simulation

When someone wants to make experiments in the mobile robots research area, two alternatives are possible: to accomplish the experiments directly in a real robot; or to make experiments using a simulated robot. The use of a real robot is more realistic, but the simulation have the following advantages[26, 15]:

- When using simulated robots, doesn't exist the risk of robot damages;
- Tasks as the exchange and recharge of batteries are not

necessary;

- The robot positioning in order to restart a simulation can be accomplished automatically, without human intervention;
- The simulation clock can be accelerated, reducing the total amount of spent time for learning;
- Several different architectures and robot models can be tested before to do an expensive practical construction of the robot. In this way it is possible to test and select the best robot model to construct.

For these reasons, we chose to implement our initial experiments using a simulated robot, through the implementation of a very realistic robot simulator, using a physical simulation engine, so we can build simulated robots very similar to the real models.

2.1 Physics simulation engine

In order to do more realistic mobile robots simulation, several elements of the real world should be present in the simulated model, doing the simulated bodies to behave in a similar way related to the reality. Especially, it is necessary that the robot suffers from instability and falls down if badly positioned and controlled, and also it should interact and collide against the environment objects in a realistic manner. To accomplish that, it is necessary to model the physics laws in the simulation environment (e.g. gravity, inertia, friction, collision). Nowadays, several physics simulation tools exist used for the implementation of physics laws in simulations. After study different possibilities, we chose a widely adopted free open source physics simulation library, called Open Dynamics Engine - ODE[1].

ODE is a software library for the simulation of articulated rigid bodies dynamics. With this software library, it's possible to make autonomous mobile and legged robots simulations with great physical realism. In ODE, several rigid bodies can be created and connected through different types of joints. To move bodies using ODE, it's possible to apply forces or torques directly to the body, or it is possible to activate and control angular motors. An angular motor is a simulation element that can be connected to two articulated bodies, which have several control parameters like axis, angular velocity and maximum force. With these elements, it's possible to reproduce articulations present in real robots, humans or animals, with a high precision level.

2.2 Modeled robot

In order to maintain the simulation speed in an acceptable rate, we should use few and simple objects. For this reason, the simulated robot was modeled with simple objects, as rectangles and cylinders, and they have only the necessary articulations to perform the gait. Thus, body parts as the head and the tail are not present in the modeled robot. In order to keep our robot project simple, the joints used in the robots legs just move around the z axis of the robot (the same axis of our knees), and the simulations just used robots walking in a straight line. In the near future, we plan to extend our system to accept more complex robot models and joints.

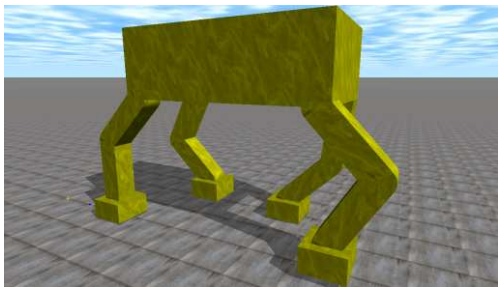

Figure 1. Robot model used

Several robot types were developed and tested, before we defined the final main model presented here, that is shown in Figure 1. The Table 1 shows the dimensions of the robot in centimeters. The simulated robots dimensions are approximately the dimensions of a medium sized dog. The joint restrictions used in the simulated robot are similar as they biological equivalent, as shows the Table 2.

Table 1. Dimensions of the robot

	Dimensions		
Part	**x**	**y**	**z**
Body	45.0cm	15.0cm	25.0cm
Thigh	5.0cm	15.0cm	5.0cm
Shin	5.0cm	15.0cm	5.0cm
Paw	8.0cm	5.0cm	9.0cm

Table 2. Joints restrictions

	Hip		Knee		Ankle	
Legs	*min*	*max*	*min*	*max*	*min*	*max*
Front	-30°	90°	-120°	0°	-15°	60°
Rear	-90°	30°	0°	120°	-60°	15°

3 Optimization methods

This section describes the optimization methods used in this work, which are neural networks, used to gait

[1] ODE – http://www.ode.org

control, and Powell's method, used to calculate the inverse kinematics and to generate the learning database generation.

3.1 Artificial neural networks

Through the use of an abstract and simplified model of human neurons, is possible to develop a neural simulator capable to classify, to generalize and to learn how and approximate functions[10]. One of the most used neural learning models is the so called Multi-Layer Perceptron (MLP) with Back-propagation learning algorithm[25]. Some improved versions of the original Back-Propagation algorithm were developed in the few past years, and the RPROP algorithm[24] become an interesting choice among them.

The RPROP algorithm performs a direct adaptation of the weight step (learning rate) based on local gradient information. To achieve this, each weight has its individual update value Δ_{ij}, which solely determines the size of the weight update. This adaptive update-value evolves during the learning process based on its local sight of the error function E, according to the following learning-rule[24]:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ * \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \Delta_{ij}^{(t-1)}, & \text{else} \end{cases} \tag{1}$$

where $0 < \eta^- < 1 < \eta^+$, $\frac{\partial E}{\partial w_{ij}}$ is the partial derivative of the error function for the weight w_{ij}, and $\Delta_{ij}^{(t-1)}$ is the last weight update.

The network topology used in this work was an Elman network, which are multi-layer back-propagation networks with a feedback connection from the output of the hidden layer to its input. This feedback path allows Elman networks to learn to recognize and generate temporal patterns, as well as spatial patterns[7]. The Figure 2 shows a diagram of the Elman networks.

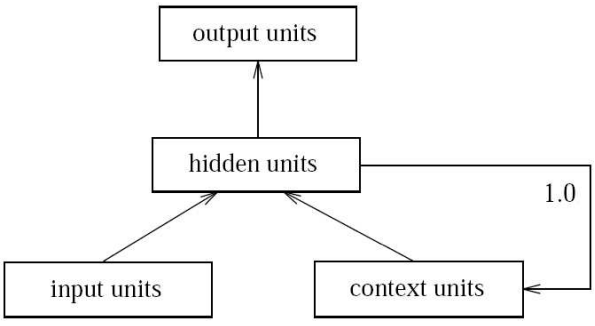

Figure 2. Diagram of a Elman network

3.2 Powell direction set method

Powell's gradient-free methods[20] provide directions of search that line up with directions conjugate about the Hessian matrix of the quadratic approximation to the objective function. The Powell method applied in the current study is the version described in[21, 1], in which an initial guess and a set of independent search directions are provided to the algorithm. In each iteration the method serially performs a sequence of line minimizations along the various directions in the space of parameters. At the end of each iteration the method replaces one of the original directions with the line joining the starting and ending points. The Figure 3 illustrates the Powell method. Care is taken to ensure that the directions remain linearly independent. The iteration is terminated when either the convergence rate or the error between the predicted and the exact solutions are smaller than prescribed values. The Powell's method described above will converge to the minimum of a quadratic function in a finite number of iterations[4].

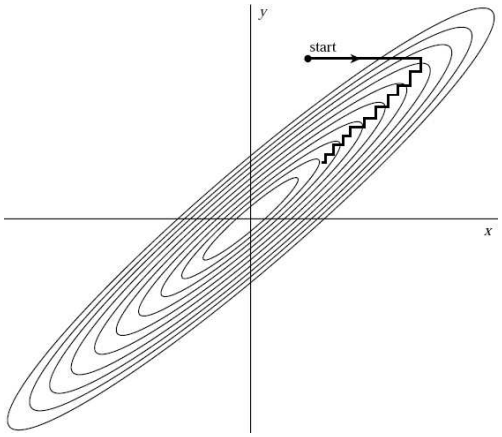

Figure 3. Powell's method[21]

4 Half ellipse control

Initially, the robot gait control was accomplished modeling the endpoints trajectory through a cyclical function, specifically a half ellipse. The Figure 4, extracted from[9], illustrates the half ellipse used. For the trajectory generation, the endpoints positions in the x axis are obtained through the equation:

$$x(t) = \begin{cases} x_0 - \frac{l}{2} + \frac{2lt}{T} & \text{if } t < T/2 \\ x_0 + \frac{l\cos(\alpha)}{2} & \text{if } t \geq T/2 \end{cases}, \tag{2}$$

where $x(t)$ is the endpoint position at time t, x_0 is the ellipse origin (center) in the x axis (see Figure 4), T

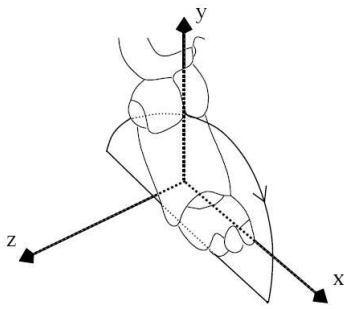

Figure 4. Half ellipse trajectory[9]

is the cycle time (one complete step), t is the current time, l is the ellipse length and α is the current endpoint angle in relation to ellipse center, calculated through the equation:

$$\alpha = \frac{2\pi}{T}\left(t - \frac{T}{2}\right). \quad (3)$$

The endpoint position in the y axis is obtained through the equation:

$$y(t) = \begin{cases} y_0 & \text{if } t < T/2 \\ y_0 + h\sin(\alpha) & \text{if } t \geq T/2 \end{cases}, \quad (4)$$

where $y(t)$ is the endpoint position at time t, y_0 is the ellipse origin in the y axis, and h is the ellipse height (see Figure 4). When $t >= T$, the current time t is reseted to 0 and a new robot step starts.

The ellipse parameters using in the above equations are optimized using the Powell's method[4]. The Table 3 shows the parameter values used in our simulations. The positions x_0 and y_0 are in relation of the origin (hip) of the leg.

Table 3. Ellipse parameters

Parameter	Value
T	4.00 seconds
Δt	0.05 seconds
x_0	3.50 centimeters
y_0	-30.00 centimeters
l	15.00 centimeters
h	8.50 centimeters

After the endpoints coordinates generation, the inverse kinematics was calculated using the Powell's method, to obtain the expected angles of each joint. In order to control the joints, the torque applied to each joint angular motor was calculated by[3]:

$$\tau_{t+1} = \max(I(\omega_t - k(\theta - \theta_d)), \tau_{max}), \quad (5)$$

where where θ is the actual joint angle, θ_d is the desired joint angle, τ_{max} is the maximum torque ceiling, $\omega = \dot{\theta}$ (joint angular velocity), and I is the inertia matrix.

The generated gait using this method was efficient, but the resulting gait control behavior has shown to be few robust in non expected situations. For these reasons, we decided to use a neural network inspired in the Elman-net model to implement the gait control, as described below.

5 Neural control

The first step in the neural gait control was to create the examples data set to be employed in the ANN learning. The half-ellipse controller, described above, was adapted in order to generate a log file, containing records of: the current joint angles at time t, the sensors state (bumpers below the paws indicating the leg phase – 0 = swing phase; 1 = sustained phase), and the desired joint angles for the next time $t+1$. In the near future we plan to use motion capture devices to provide the examples for the training database.

The learning database was created with 6000 examples, (3000 for the learning and 3000 for the generalization test), each one with 16 inputs and 12 outputs. The first 12 inputs are the current joint angles, and the 4 last inputs are obtained through bumpers. The 12 outputs are the expected joint angles at time $t + 1$. With the neural network controller, there is no need to calculate the inverse kinematics. The neural network used in the experiments was an Elman network, as shows the Figure 5. This figure is just illustrative, because we used in our experiments more neurons and connections per layer.

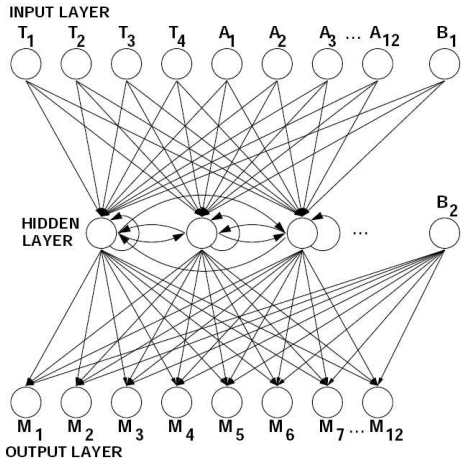

Figure 5. Example of an Elman network[3]

The Artificial Neural Network simulator used in our experiments was the Stuttgart Neural Network Simulator - SNNS[2], it is a free software, and a quite complete

[2]SNNS – http://www-ra.informatik.uni-tuebingen.de/SNNS/

neural network simulator that have several additional tools that allow us to create scripts and execute learning and simulation tasks in batch mode. The SNNS facilities also simplify the analysis of the obtained results and creation of graphic plots.

The Table 4 shows the main ANN parameters used in our simulations. For a complete description of these parameters, see the SNNS manual. The input layer is fully

Table 4. Neural network parameters

Parameter	Value
ANN model	MLP - Elman
Learning algorithm	RPROP
Number of input units	16
Number of hidden units	10
Number of output units	12
Activation function input	Act_Identity
Activation function hidden	Act_TanH
Activation function output	Act_IdentityPlusBias
Starting learning rate	0.001
Maximum learning rate	0.01
Weight decay	1.0×10^{-5}
Teacher forcing	0.0
Weights initialization	$[-0.0001; 0.0001]$
Initial activation (context)	0.2
Maximum of generations	1000

connected to the hidden layer, and the hidden layer is fully connected to the output layer. In addition, the hidden layer is fully, recurrently connected (Elman architecture). At each time step of the simulation of a robot's behavior, the eight sensor signals are scaled to floating-point values in $[-1.0; 1.0]$ interval, and supplied to the input layer. The values are propagated to the hidden and output neurons. The hidden neurons use the hyperbolic tangent activation function.

6 Results

The Table 5 shows the results obtained in the simulations. The first column (**E**) is the experiment identifier, the second column is the mean square error (MSE) in the learning database, and the last column is the MSE in the generalization test database (of the best epoch). The generalization test was performed each 10 epochs. The last three lines shows the mean (μ), the standard deviation (σ) and the 95% confidence interval (**CI**) of the five accomplished experiments. The Figure 6 shows the neural output error evolution curve during the leaning related respectively to the learning and the generalization test databases.

Table 5. Obtained results

E	MSE Learning	MSE Generaliz
1	0.0104	0.0168
2	0.0094	0.0194
3	0.0124	0.0174
4	0.0119	0.0199
5	0.0086	0.0174
μ	**0.0106**	**0.0182**
σ	**0.0016**	**0.0014**
CI	$[\mathbf{0.008}; \mathbf{0.012}]$	$[\mathbf{0.016}; \mathbf{0.019}]$

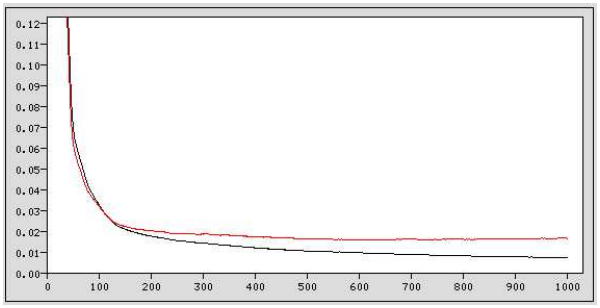

Figure 6. Progress of the neural learning

Observing the gait behavior of the robots trained using neural networks, they are much more stable and robust than the gait behavior when controlled directly by the half-ellipse, because the neural network uses sensor information (bumpers) and have the capacity to generalize to non expected situations. The Figure 7 shows the

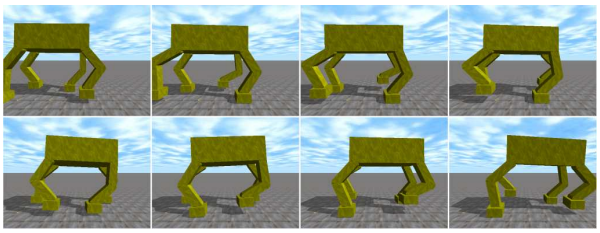

Figure 7. Half-ellipse robot gait

gait control generated using locus based gait, and The Figure 8 shows the gait control generated using the neural network[3].

7 Conclusions and perspectives

The main goal of this paper was to describe the use of a gradient descent learning algorithm (RPROP) to adjust weights of a recurrent neural network used in a legged

[3]Some videos of the learned gait control are available in the site http://www.inf.unisinos.br/˜osorio/leggen

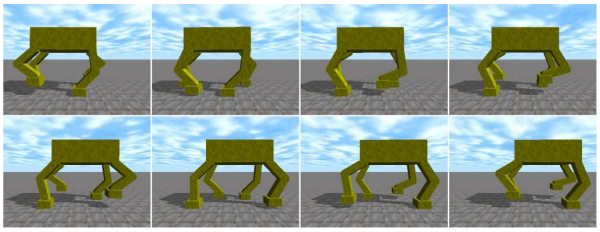

Figure 8. Neural network robot gait

robot gait control. For this, it was described the properties of the neural network used and the methodology for the generation of the examples database used in the neural learning. The neural networks results are more robust than the results obtained by the controller based in the locus based functions, and the ANN has the advantage that it does not require the inverse kinematics calculation.

The perspectives of this work includes to use motion capture devices to generate the learning database files, and to use more complex robots, like bipeds. We want too use more sensor information (gyroscope, accelerometers, sonars) to improve the quality of the obtained gaits. The perspectives also includes to adapt gait control in order to make possible control robots moving over irregular surfaces and to climb or to descend stairs.

References

[1] F. S. Acton. *Numerical Methods that Work*. Harper and Row, New York, 1970.

[2] G. A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. MIT Press, Cambridge, MA, 2005.

[3] J. C. Bongard and R. Pfeifer. A method for isolating morphological effects on evolved behaviour. In *Proc. 7th Int. Conf. Simulation of Adaptive Behaviour (SAB)*, pages 305–311, Edinburgh, UK, Aug. 2002. MIT Press.

[4] R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[5] J. Busch, J. Ziegler, C. Aue, A. Ross, D. Sawitzki, and W. Banzhaf. Automatic generation of control programs for walking robots using genetic programming. In *Proc. European Conf. Genetic Programming (EuroGP)*, pages 258–267, Berlin, Germany, 2002.

[6] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge Univ. Press, Cambridge, UK, 2000.

[7] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[9] D. Golubovic and H. Hu. Ga-based gait generation of sony quadruped robots. In *Proc. 3th IASTED Int. Conf. Artificial Intelligence and Applications (AIA)*, Benalmadena, Spain, Sept. 2003.

[10] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, Upper Saddle River, NJ, 2 edition, 1999.

[11] M. R. Heinen and F. S. Osório. Applying genetic algorithms to control gait of physically based simulated robots. In *Proc. IEEE Congr. Evolutionary Computation (CEC)*, Vancouver, Canada, July 2006.

[12] M. R. Heinen and F. S. Osório. Gait control generation for physically based simulated robots using genetic algorithms. In *Proc. Brazilian AI Symposium (SBIA), to appear*, LNCS, Ribeirão Preto, SP, Brazil, Oct. 2006. Springer.

[13] D. Jacob, D. Polani, and C. L. Nehaniv. Legs than can walk: Embodiment-based modular reinforcement learning applied. In *Proc. IEEE Int. Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 365–372, Espoo, Finland, June 2005.

[14] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 2619–2624, New Orleans, LA, Apr. 2004.

[15] A. M. Law and D. W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, 2000.

[16] M. A. Lewis, A. H. Fagg, and A. Solidum. Genetic programming approach to the construction of a neural network for control of a walking robot. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 2618–2623, Nice, France, 1992.

[17] R. B. McGhee. Robot locomotion. *Neural Control of Locomotion*, pages 237–264, 1976.

[18] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[19] T. Mitchell. *Machine Learning*. McGrall-Hill, New York, 1997.

[20] M. J. D. Powell. An efficient method for finding the minimum for a function of several variables without calculating derivatives. *The Computer Journal*, 7:155–162, 1964.

[21] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge Univ. Press, Cambridge, MA, 1992.

[22] M. H. Raibert. *Legged Robots That Balance*. MIT Press, Cambridge, MA, 1986.

[23] R. Reeve and J. Hallam. An analysis of neural models for walking control. *IEEE Trans. Neural Networks*, 16(3):733–742, May 2005.

[24] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. IEEE Int. Conf. Neural Networks (ICNN)*, pages 586–591, San Francisco, CA, Mar. 1993.

[25] D. Rumelhart, G. Hinton, and R. Williams. *Learning Internal Representations by Error Propagation*. MIT Press, Cambridge, MA, 1986.

[26] K. Wolff and P. Nordin. Evolutionary learning from first principles of biped walking on a simulated humanoid robot. In *Proc. Advanced Simulation Technologies Conf. (ASTC)*, Orlando, FL, Apr. 2003.