Developing Autonomous Robot Controllers using Genetic Algorithms

Robert Rose

October 2, 2004

Abstract

This paper presents a summary of experiments conducted to evolve autonomous robot controllers that can balance a physically simulated articulated character.

1 Introduction

Motion capture offers a quick solution to provide realistic human motion for animation applications. Human body motion can be recorded and "played back" by a computer to mimic the body that was recorded. Working with motion capture data however is cumbersome: transitioning between sequences, modifying the motion, and scaling the motion to a different-sized character are all difficult tasks.

A new area in computer animation that is growing in popularity is integrating motion capture movement with physically simulated movement. "Rag-doll physics" used in many video games (Unreal Tournment, Half-Life, etc.) involve using motion capture for the "living" character animation and then transitioning to a physically simulated character when the character "dies," to give a more life-like feel to the character's suffering when their health meter goes below zero. Real-time back-and-forth transitioning between physically simulated movement and motion capture movement is an area the author has yet to see in computer animation, for example, transitioning to physically simulated movement every time the character takes damage, rather than only when they die.

A far more realistic approach to generating human character motion for computer animation could be to completely physically simulate *all* of the motion. Physical simulation of all character movement requires development of bones, joints, skin, muscles, and a "virtual brain" to accurately control

the character. This approach has been tried with varying degrees of success by several groups over the last few years [1] [2]. Previous attempts have used genetic algorithms to evolve a neural network to control muscle movement. Other work has evolved even the creature itself [3].

This paper presents a summary of experiments conducted to evolve autonomous robot controllers that can balance a physically simulated articulated character. The controllers are based on a neural network who's weights and structure is evolved through genetic programming. In all experiments the size and structure of the character was fixed. The character will live if a physically simulated environment using the Open Dynamics Engine[1].

2 Experimental Setup

2.1 Character Model

The character was modeled using three limbs connected by two hinge joints, the bottom limb being oriented horizontally with the "floor" and the two other limbs oriented vertically, mimicking the structure of a human leg. Limbs were modeled in 3-dimensions as rectangular blocks and given weights proportional to those of a human (approximated). A 2-dimensional side-view of the character model is given in Figure 1.

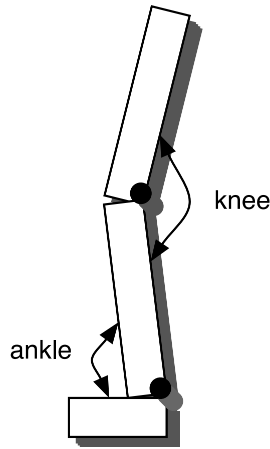

Figure 1: Character Model (2D side-view)

The character model was given the following inputs:

[1] http://opende.sourceforge.net/

- The height of the character as determined by observing the position of the top limb segment (the "thigh").
- The angles of the "knee" and "ankle" joints.
- Whether or not the character is touching the ground.

The character model was allowed two outputs: torques to be applied at the "knee" and "ankle" joints.

2.2 Physical Simulation

The simulation was written in C++ with the physical environment simulated using the Open Dynamics Engine (ODE), an open source library for simulating rigid body dynamics.

The character when placed in the environment is simulated by ODE. Several shortcuts were taken to speed the simulation and to make implementation easier:

- Few external forces. The character is only effected upon by gravity, $(-9.8m/s^2)$.
- Few external collisions. The character may only collide with the "floor" which is located at $y = 0$.
- No internal collisions. Inter-penetration within the character is not considered. Instead, joint limits are placed on the character's joints to prevent limbs from excessively intersecting.
- Simplified contact joints. When any part of the character comes in contact with the floor ODE contact joints are used to "bounce" the character away from the floor. ODE has many contact joint parameters available to model contacts between two bodies, however for the purposes of simplifying the simulation and to improve stability the only contact joint parameter used was `dContactBounce`.
- "Large" objects. Floating-point errors occur with ODE if "small" objects are used, so the characters were made larger than necessary to help prevent jitter. Originally, characters of height `1.0f` were used, but it was discovered that collision detection was unstable at this scale. Increasing the size of the character to `10.0f` resolved the collision jitter issues.

- Larger CFM, smaller ERP. ODE uses two constants to control joints and collisions, the Constraint Force Mixing (CFM) parameter and Error Reduction Parameter (ERP). These values are used to scale corrective forces when bodies collide and when joints begin to separate. It was found that for the purposes of this simulation the default values enforced too "rigid" of collisions, so the CFM was increased to `0.0001f` (default is `0.00001f`), and the ERP was decreased to `0.1f` (default is `0.2f`). These "softer" values allowed more natural and forgiving behavior for the robot controllers.

2.3 Character Controller

At each time step the character was given an opportunity to inspect it's current inputs (see above) and generate torques to be applied at it's joints. This behavior was provided using a simple two-level recurrent neural network, as given in Figure 2. The neural network was assigned weights for the inputs to all nodes and the sigmoid function was used to determine the output value for each node. It was determined through experimentation that an appropriate range of values for the weights of the neural network was between -5.0 and 5.0.

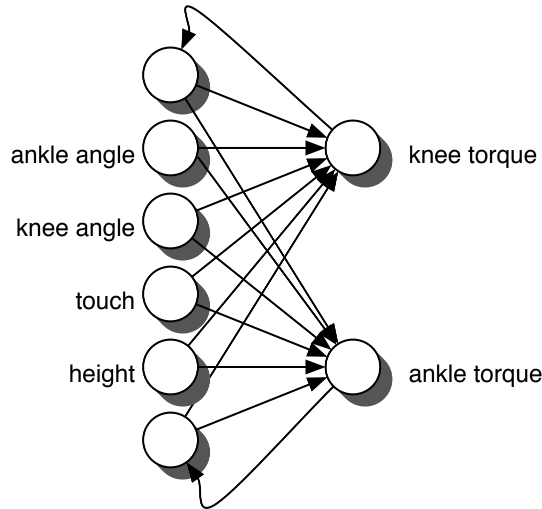

Figure 2: Character Controller Neural Network

The process for using the neural network was the following:

1. Map the inputs (knee angle, ankle angle, height, etc.) to different

ranges.

2. Allow the neural network to propagate for one time step.

3. Map the outputs (knee torque, ankle torque) to different ranges.

2.4 Chromosome Representation

To perform evolution, each robot controller must be represented as a "chromosome". For these experiments the chromosome representation of the robot controller was an array of the node weights from the controller's neural network. Weights when represented in a chromosome are referred to as "genes." Each weight, while used by the neural network as a floating point value, has an equivalent representation in the chromosome as an unsigned integer. The mapping between these two representations is as follows:

$$f = (i/i_{max}) * f_{max} - f_{offset} \tag{1}$$

Where f is the floating point representation, f_{max} is the maximum desired weight, f_{offset} is an offset of the maximum desired weight, i is the unsigned integer representation and i_{max} is the maximum unsigned integer representation.

It was useful for the purposes of the experiment to randomly generate a chromosome. This was accomplished by creating an array of random unsigned integers, then mapping them to their floating point representations, and then assigning the results to the weights of the controller's neural network.

2.5 Fitness Evaluation

In order to determine what the best controller is for a given simulation run a "fitness function" that rates the performance of the controller is necessary. For the objective of the balancing controller, the fitness function should measure how well the controller balances the character. For example a fitness function that could be used for this objective simply measures the height of the character at the end of the simulation.

Depending upon how the fitness function is written, strategies for meeting the objective may emerge that are not what is expected. To discourage undesirable strategies, two different approaches to the fitness function were taken to meet the balancing objective:

1. $f(t) = h(t)$, where $h(t)$ is the height of the character at time t.

2. $f(t) = f(t-1) + b(t) - c(t)$, where $b(t)$ is a "benefit" reward for how well the controller is doing at time t, and $c(t)$ is a "cost" penalty for the controller behaving poorly or acting in an undesirable manner at time t. The simulation was initialized with $f(0) = 0$.

The functions $b(t)$ and $c(t)$ were arrived upon through experimentation and observation of the behaviors of the robot controllers:

$$\begin{aligned} b(t) &= h(t) * 0.1 + 0.01 * t \\ c(t) &= -5.0\ if\ contacts > 0 \\ &= 0.0 \text{ otherwise} \end{aligned}$$

Where *contacts* is the number of contacts the robot is making with the floor. The motivation for the recurrent fitness function was to encourage controllers that might attempt to balance themselves but eventually fall over, the belief being that they could be evolved into controllers that can reliably balance.

2.6 Evolution

The process of evolving the robot controllers was done in a style similar to techniques presented by Nolfi and Floreano [2]. Evolution begins with a large population of randomly generated chromosomes which are mapped to their corresponding robot controllers (see above). The fitness of each controller is evaluated, and the "fittest" robots are chosen for reproduction. A generation of offspring is then created, their fitness is evaluated, and the process repeats until a "fit enough" robot controller is found.

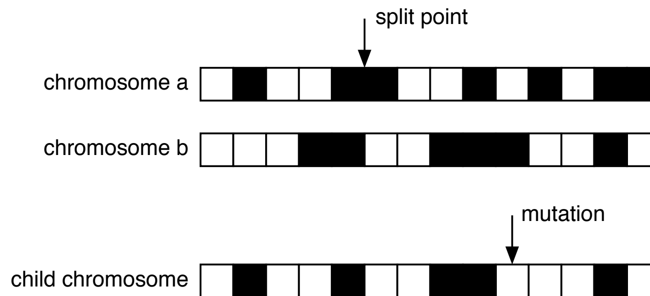

Figure 3: Chromosome Combination and Mutation with Two Parents

The offspring are created via combination and mutation of their parent(s), as shown in Figure 3. For the purposes of this experiment, combination was decided randomly to be of one or two parents, allowing for both asexual and sexual reproduction. Combination of one parent merely cloned the parent to make a child. Combination in sexual reproduction occurs by picking a random chromosome "split point" and making the child offspring's chromosome the first parent's chromosome before the split point and the second parent's chromosome after the split point.

After combination (or copying if reproduction is asexual) the child's chromosome is then randomly mutated. Mutation occurs by sampling one or more values in the child chromosome and modifying them randomly. Several styles of modification are possible:

- Flip one or more bits in the chromosome randomly.
- Choose random gene(s) in the chromosome and add one or more random numbers to it.

Adding one or more random numbers to a gene in the chromosome is an idea borrowed from Sims [3], which allows small changes in the controller's behavior to be more likely than dramatic changes. Mutation of this style requires you specify a range of random numbers to add to the gene and the number of genes to be modified at this range. It is possible to specify more than one set of ranges and numbers of genes.

Mutation of Sims' style was chosen for this experiment. Three different forms were tested:

1. 10 genes with range [-100, 100], then 2 genes with range [-500, 500], finally 1 gene with range [i_{min}, i_{max}].
2. 5 genes with range [-1000, 1000], then 1 gene with range [i_{min}, i_{max}].
3. 1 gene with range [i_{min}, i_{max}].

Where i_{min} and i_{max} are the minimum and maximum values of integers representable by a gene in a chromosome.

2.7 Optimizations

Two optimizations were used to improve the speed of evolution:

1. Early pruning. If it was determined that a controller's fitness was too low and that there was little hope for recovery, the controller was stopped to allow more CPU time to be given to other controllers.

2. Less rendering. Rendering the scene of characters during evolution isn't necessary for evolution to occur, so rendering was only omitted during much of the evolutionary process.

3 Results

3.1 Evolved Strategies

Three common strategies were observed during evolution of the controller that successfully met the criteria set forth by the fitness functions (Figure 4). The first strategy, labeled A in the diagram, was the "optimal" strategy where the controller kept itself balanced by applying fine corrective forces to the knee and ankle joints. With strategy B the controller applied maximal forces to the knee and ankle joints, compressing the character, which happened to keep it balanced. Strategy C was a complete surprise; with this strategy the character compressed but then "popped" itself up to balance on the end of it's toes.

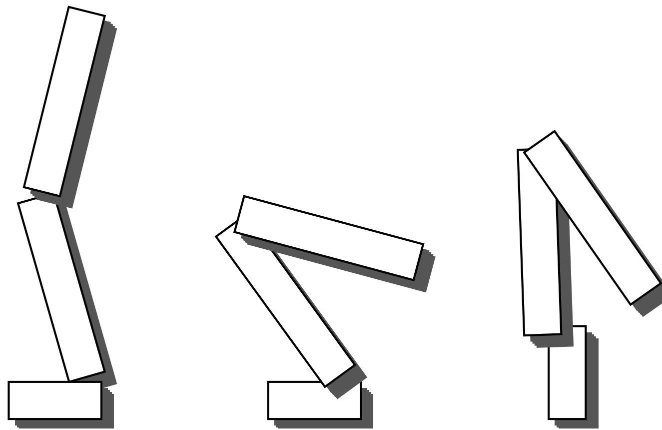

Figure 4: Commonly Evolving Strategies (A, B and C)

Initially strategies A, B, and C were the only observed strategies, and strategy A was believed to be the most desirable. Because of this belief, work on evolving a controller for the character was focused on encouraging the development of controllers that followed strategy A. Fitness Function 2

was actually written to encourage the development of strategy A–the belief being that controllers that could stand for a brief moment could be mutated into controllers that could reliably stand. Unknowingly however, writing Fitness Function 2 created two new less-common strategies, D and E, given in Figure 5.

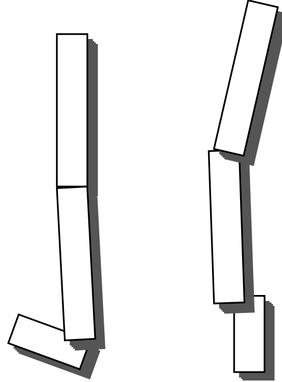

Figure 5: Less Commonly Evolving Strategies (D and E)

Each strategy had an approximate height that it was capable of balancing the character at. These approximates are given in Table 1. When viewing the figures at the end of this paper you can use this table to correlate the height to a strategy.

Table 1: Approximate Heights Attainable by Strategy

Strategy	Height
A	15.5
B	9.4
C	12.3
D	19.5
E	21.7

3.2 Trials

Testing the evolutionary system was done in trials of 3 independent variables: number of controllers in each generation (10, 100, or 1000), mutation technique used (1, 2, or 3), and fitness function (1 or 2). The number of

parents used to create offspring was fixed for all experiments at 10. Trials of 1000 controllers over 20 generations took approximately three hours on an Athlon XP 2600 computer.

The figures shown at the end of the paper summarize the different trial groups. In each figure the x axis is the generation (each trial including 20 generations) and the y axis is the height attained by the controller that best met the fitness function. The legend indicates the number of controllers used in each generation ("1000a" or "1000 agents"), the number of parents used to create offspring ("10p" or "10 parents"), and the mutation technique used ("1m" or "mutation technique 1").

Figure 6 shows 8 trials using fitness function 1 with all 3 mutation techniques. Only mutation techniques 2 and 3 were able to discover strategy A using fitness function 1.

Figure 7 shows the results of 6 trials using mutation technique 1 with fitness function 2. The trial of generation size 10 failed to produce any successful strategy. The other generation size trials discovered strategies A, B, and C but were unsuccessful mutating to strategies D or E.

Figure 8 shows the results of 5 trials using mutation technique 2 with fitness function 2. Several trials were successful at finding strategy B, but eventually moved on to strategies A or C. One trial, displayed in yellow, was able to discover strategy A and then attempt to move to strategies D and E.

Figure 9 shows 4 trials using mutation technique 3 with fitness function 2. All trials optimized strategies A or B. Only one trial was able to make the leap from strategy B to strategy A.

Complete data for each trial, including the average and standard deviation of the controllers performances, is available on the website for this paper.

3.3 Discussion

Often times the evolution of the robot controllers in this experiment appeared to be doing nothing more than hill-climbing. In many of the trials it appeared that once the controllers found a successful strategy they merely optimized that strategy rather than attempt a dramatic change to find a new strategy.

One of the trials however did seem to have the right mutation and fitness function settings to attempt new strategies: Mutation technique 2 with fitness function 2. This combination of settings appeared to have the right balance of a) encouraging optimization of existing strategies and b) dramatic

mutation to attempt new strategies.

The "benefit-cost" fitness function 2 was clearly a winner as strategies D and E would have probably never developed without it. Writing the fitness function as a benefit-cost system allowed controllers to attempt new strategies and fail, but get another shot at it in the next generation if the strategy was showing promise.

It is quite possible that the difficulty each controller faced "leaping" to a new strategy could be caused by the simplicity of the neural network used. Further experimentation is necessary to determine if more complex (such as two and three hidden layer) neural networks could allow more complicated strategies and more successful mutations. Encoding the structure of the neural network in the chromosome could also prove useful, allowing mutations of not only the weights but the neural network itself. This will definitely be the basis of further research!

4 Conclusion

Evolutionary robotics holds much promise for developing autonomous robot controllers, but the evolutionary utopia your author previously believed in has been shattered by this experiment. Developing a robust simulation environment and fine-tuning the evolutionary parameters such as the amount of mutation and the fitness function is not an easy task and requires much human intervention to be successful. Evolutionary robotics is difficult.

References

[1] R. Smith, "Intelligent motion control with an artificial cerebellum," Ph.D. dissertation, University of Auckland, Auckland, New Zealand, 1998.

[2] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press, 2000.

[3] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques.* ACM Press, 1994, pp. 15–22.

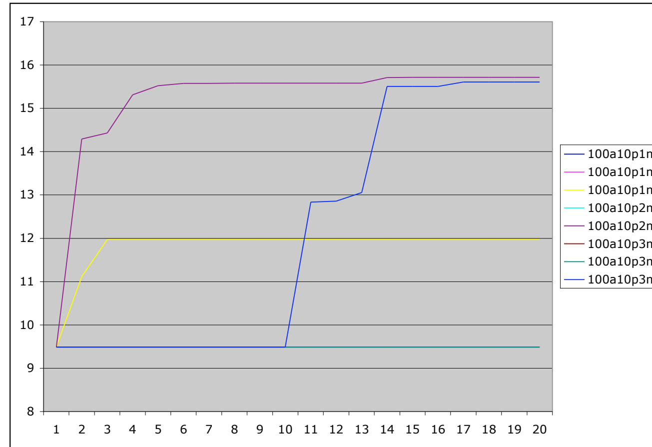

Figure 6: Fitness function 1

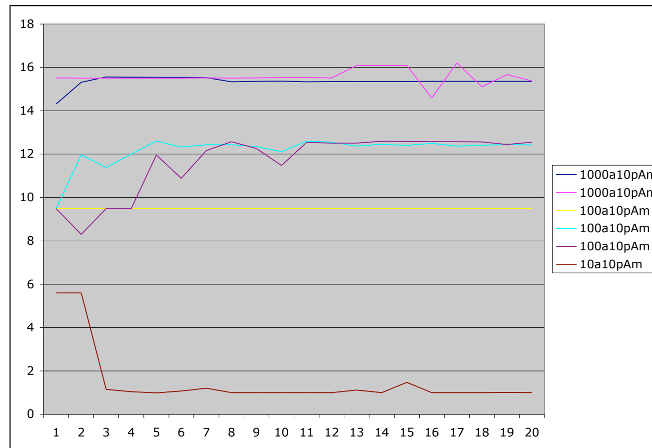

Figure 7: Mutation technique 1 with fitness function 2

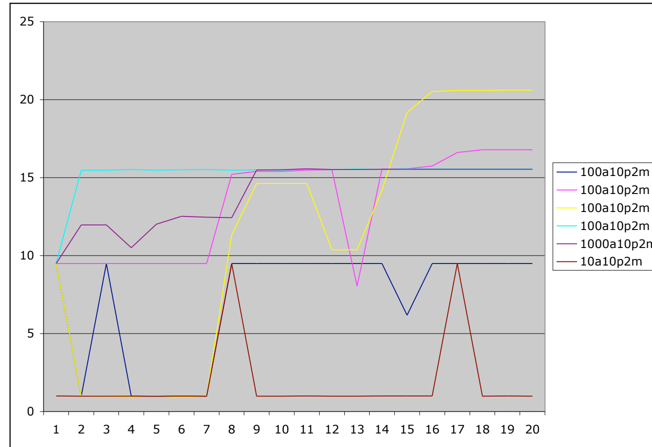

Figure 8: Mutation technique 2 with fitness function 2

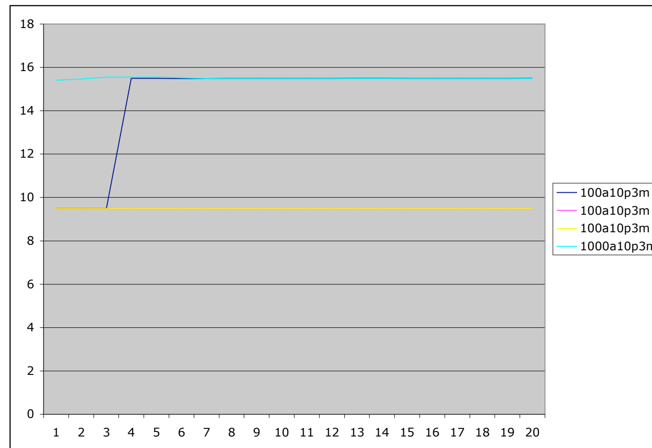

Figure 9: Mutation technique 3 with fitness function 2