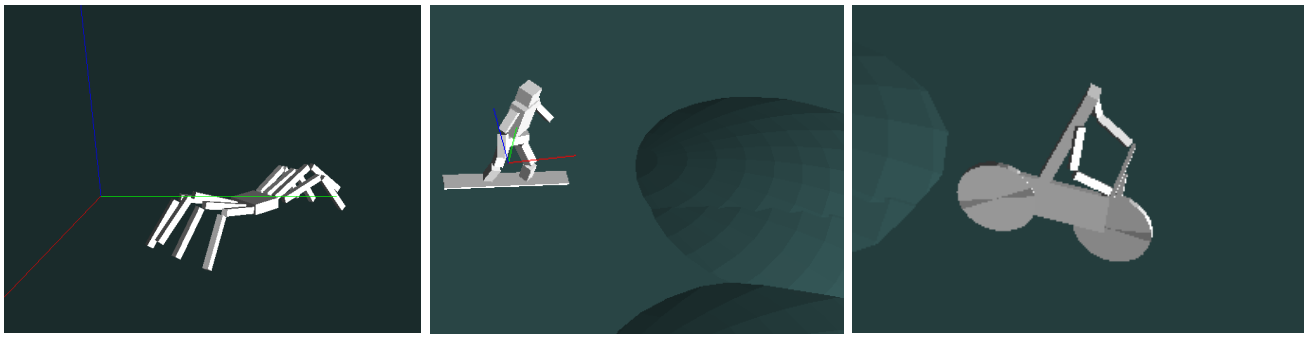
Virtual Character Locomotion using Simulated Annealing

Eduardo Poyart – UCLA – March 2010
CS 275 – Artificial Life – Prof. Demetri Terzopoulos



Abstract

In this work, low level control of three different virtual characters is achieved using simulated annealing to make these characters perform locomotion. Aspects that were studied include joint positioning and their characteristics (including degrees of freedom), control scheme, neural network inputs and outputs and annealing schedule. A visual annealing approach optionally allows the experimenter to observe the annealing process animated as it happens. A modeling pipeline based on Blender provides a sofisticated way to model characters, and the character loading and assembly in the runtime is data-driven. Three characters were created and used: a spider, a snowboarder and a cyclist. Good results were obtained with the spider and the cyclist; they successfully learned to locomote on the virtual terrain.

1. Introduction

Animating virtual characters using dynamic control under physics is a very attractive possibility. It allows those characters to respond better to unforeseen events on the environment, as compared to predefined, fixed animations. Even more interesting is teaching those characters using learning approaches such as simulated annealing or genetic algorithms.

In this work, I employ simulated annealing to teach virtual characters to locomote on the environment. Joint control is done either by frequency-domain controllers or neural networks in different characters. The characters that were created and used for this project were a spider, a snowboarder and a cyclist. Rather than making the annealing process completely opaque, I used a *visual annealing* approach. Visual annealing refers to the fact that the animation can be seen in real-time during the annealing process, if that is desired. This can provide important clues as to how the character is behaving during its learning.

Related work is presented in section 2. In section 3, Locomotion, I classify the characters with respect to their locomotion characteristics. In section 4, Simulation System, I show the main aspects of the simulation system that was developed for this project, as well as its two open-source components: the physics simulation library Open Dynamics Engine and the modeling software Blender. The annealing schedule is described in section 5. In sections 6, 7, and 8, the three virtual characters are described in detail. Results are discussed in section 9, and section 10 is the conclusion.

2. Related Work

The main inspiration for this work came from [Grzeszczuk and Terzopoulos 1995], in which sharks and other animals are modeled using mass-spring systems and learn to swim through simulated annealing. The method of simulated annealing, described in detail in [Kirkpatrick, Gelatt and Vecchi 1983], is a global optimization method that is capable of finding parameters in a high dimensional space that optimize an objective function. It was used here to find motor controller parameters for all three virtual characters. In [Grzeszczuk, Terzopoulos and Hinton 1998], a neural network is used for animation control of physics-based models. The basic ideas of each of those three works are combined here. In [Hodgins et al. 1995], many human characters perform different types of athletics, including bicycle riding. In that work the actuation parameters were manually set up.

3. Locomotion

Virtual characters can be classified according to their stability. The spider is a stable character: it does not fall or flip over when left alone in the simulated world. The snowboarder and cyclist are unstable: they need active control in order to stay upright. Another classification scheme, which is orthogonal to this one, is related to whether the character locomotes with cyclical control. The spider needs to move its legs in a certain cyclic way. A snowboarder does not need cyclic motion if we think of him as just sliding down a track; what he needs to do is maintain balance and respond to eventual bumps on the ground. A cyclist is a hybrid case: he needs both to cycle his legs and maintain balance. However, the model implemented in this work glides down an inclined path. He uses his potential energy and does not need to move his legs, and is therefore analogous to the snowboarder whose main objective is to maintain balance and glide down as far as possible.

In view of this, the characters can be divided into two groups. The spider is a character that needs to move its legs and doesn't need any sensory input. For this case, a frequency domain controller was used. The snowboarder and the cyclist, on the other hand, are characters that don't need cyclic motion but need sensory input in order to keep balance. To achieve sensory control, a neural network was used for the snowboarder and the cyclist instead of a cyclical motion generator.

4. Simulation system

A custom learning and simulation system was written, using the Open Dynamics Engine [ODE], an open-source software, as a physics simulation provider. ODE implements a variation on the integration scheme described by [Baraff 1997]. A modular diagram of the system is shown in Figure 1. Modeling is done in Blender, an open-source 3D modeling tool [Blender]. A custom Blender exporter, written in Python, makes the model available to be used by the system. The modeling process is explained in detail below. Depending on the model used, either a Frequency Domain Controller or a Neural Network is active controlling the model. During the annealing phase, the Annealing module drives the simulation and controls the FDC and NN parameters. During annealing, the renderer can be either activated (visual annealing) or deactivated. When deactivated, the system can be compiled on a fast non-graphical machine such as a multi-core server (although no multithreading was implemented so far). In the experiments performed at UCLA, the machine lion.cs.ucla.edu was used for non-visual annealing.

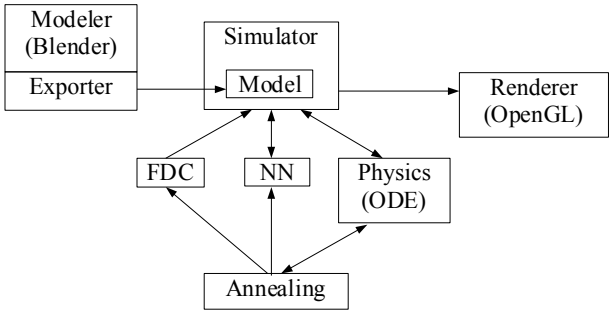


Figure 1: System components. Arrows represent information flow. FDC is the Frequency Domain Controller. NN is the Neural Network.

Once annealing is done, the system can be set up to "play back" the model using the neural network or frequency-domain controller that was obtained. During this play back phase, which is in real time, the user can interact with the model by dragging it around with the mouse. This feature proved very useful even before the annealing for the purpose of testing the various stiffness, damping and friction parameters in the model and the world. After annealing, this feature was used to prove that the model could locomote in different directions and through different areas of the terrain (with diverse obstacles). This indicated that the controllers were not overfit.

Modeling was done using the open-source software Blender. This software has an export plug-in system which consists of Python plugins that access the Blender API. There is an existing plug-in that exports to the X3D format, a standard for 3D scene description [Web 3D Consortium]. I customized this X3D exporter to include information about joints and limits. To represent a joint, a small box object can be used. Its center represents the position of the joints, and its axes are aligned with the rotation axis of the joints. More detail can be seen in [Poyart 2010], another project developed in parallel with this one. In that work, an articulated Mars rover was modeled and exported using the same system.

The modeling process is as follows. Models are created in Blender using boxes, cylinders and spheres. Joints are created with parameters that indicate the name of the objects they attach to, and with position and orientation set

according to the desired rotational axis. The model is then exported using the customized X3D exporter. A C++ class, *Model*, is the run-time representation of the model. It has the ability to read X3D files, initialize the model in ODE, render it, and expose its characteristics to other parts of the system.

This modeling pipeline was extremely useful for quickly creating models and experimenting with them. It makes the system data-driven, reducing the need to change source code when using different models. Its usefulness was proven in the Mars rover project mentioned above. Other researchers at UCLA's MAGIX lab have also expressed interest in using it. I intend to continue developing it and publish it as open-source.

5. Annealing

The annealing algorithm was based on [Kirkpatrick, Gelatt and Vecchi 1983] with some modifications. The temperature T used varied from 1.0 to approximately 0.0. It is directly related to the probability of keeping a state that reduces the fitness. Also, the maximum distance travelled by the parameter set at each step decreases with temperature, following these formulas:

$$d = r\, d_{max}$$
$$d_{max} = k\,(T + 1)$$

where d_{max} is the maximum distance travelled, r is a random number between -1 and 1, and k is an empirically determined constant. This formula means that when T = 1, d_{max} is twice as big as when T = 0. The actual change in parameters, d, is a random value between $-d_{max}$ and d_{max} . The parameters that are changed are amplitudes and phases of sine waves, in the case of the spider, and weights and biases of the neural network in the other cases. More than one parameter can be changed in each iteration. I've set the number of changing parameters to 4, in order to avoid always having movements in high-dimensional space that are parallel to one of the axis.

The other annealing parameters such as the final temperature at which the procedure ends and the rate of decrease in temperature varied from experiment to experiment, but I found I had good results in general with values around 0.001 for final temperature and 0.9998 for rate of change.

6. Spider

As a first step, a spider model was created and used. The spider model has one main body and eight legs. The legs are articulated with the main body with universal joints, which gives them two rotational degrees of freedom: one around the horizontal axis to allow the leg to move up and down, and one around the vertical axis to allow the leg to move back and forward. This gives each leg enough flexibility to be capable of propelling the spider forward. The two segments of each leg are not articulated among themselves. In order to control the spider, I chose to actuate the joints with torques, which simulates the action of muscles in the body of the animal. Later on, I switched the system to use PD controllers, as discussed below.

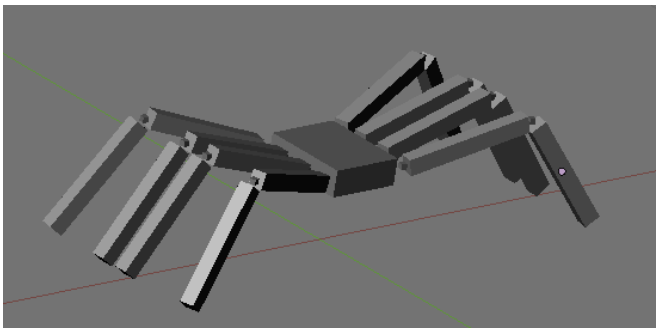


Figure 2: Spider

It seemed adequate to provide the spider with a pattern generator similar to those described in [Grzeszczuk and Terzopoulos 1995]. Specifically, a frequency-domain controller was used due to the fact that it favors periodicity, making the transitions smooth between cycles. In the work mentioned above, the authors concluded that frequency domain controllers were more difficult to anneal than time domain controllers, due to the irregularity of the topography of the objective function in high dimensional space. Here, I chose to experiment with a frequency domain controller despite its longer convergence time, mainly because of the advantage of using a representation that is natively periodical, eliminating the step of performing a Fast Fourier Transform to move from time to frequency domain.

Many attempts were made to control the spider, as well as the snowboarder, using simple torques at the joints. These attempts did not work. My initial hope was that torques coming directly from the frequency domain controllers would be sufficient to control the spider joints. This hope is more understandable in the case of the snowboarder, in which I wanted the neural network to directly control torques, hoping that a “PD controller” would emerge from the neural network. This was not achieved in either of these cases.

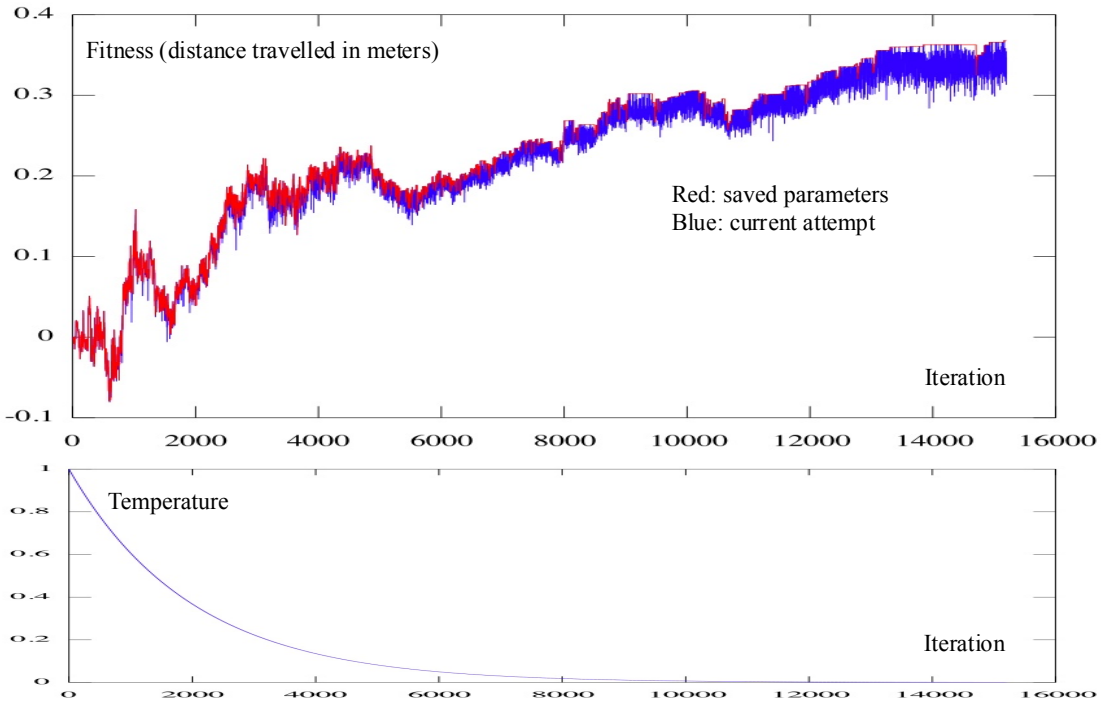


Figure 3: Spider annealing plot: fitness and temperature vs. iteration

My conclusion is that I ran into the curse of dimensionality of neural networks, described by [Grzeszczuk, Terzopoulos and Hinton 1998]. Many more neurons and hidden layers would be needed to make that kind of controller emerge. With that, I decided to take a more practical approach: to use PD controllers on the joints. I argue that it doesn't break the cleanliness and beauty of the system (contrary to what I believed before), because we can think of PD controllers as parts of our brains that are modeled separately. The analogy is as follows: the cortex sends a signal to a "PD controller" area of the brain saying that I want to position my joint wth a certain target angle. The PD controller takes care of the details, specifically what signals to send to the muscles, or what torques to send to the joints. From a high level perspective, the control signals are target angles, and that's what I modeled as outputs from my neural network and frequency-domain controller.

After a long annealing of over 16000 iterations, the spider learned a believable locomotion cycle and was able to move forward. The results can be seen in the accompanying video. Although it is not identical to a real spider, the resulting leg movement was quite interesting: upon close inspection, one notices that at each leg, vertical and horizontal movements are synchronized in such a way that it touches the ground and propels the animal forward. Figure 3 shows a plot of iteration versus objective function.

7. Snowboarder

The snowboarder model that was built in Blender is shown on Figure 4. On the run-time system, anisotropic friction was used between the board and the ground. The friction is less pronounced in the direction along the length of the board than in the direction across the board. This makes the board tend to move in the forward-back direction, rather than sideways, which mimics reality. A neural network was used to control the hip (two degrees of freedom), the knees (one degree of freedom each) and the feet (also one degree of freedom each). The inputs to the neural network are the projections of the head and the hip on the direction across the board, to indicate if the character tends to fall to the right or left.

Using PD controllers made the character surprisingly stable if there are no obstacles, even without any actuation on the joints. The character could slide downhill without falling. This was not very interesting for my purposes. In order to make the problem more interesting, I introduced bumps along the path, which were modeled as underground spheres that protrude over the surface of the ground. Tests were made with the "mesh" object in ODE, but the ODE

implementation of mesh collisions turned out to cause occasional huge forces which led to crashes. I assumed it was because of the way the board interacted with each triangle of the mesh, sometimes completely penetrating it. The “box and spheres” approach worked well without crashes.

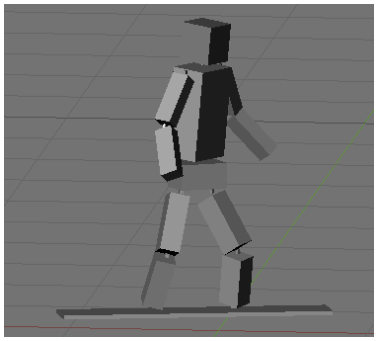


Figure 4: Snowboarder

I did not get good results with the snowboarder model. It turns out that this configuration, with the legs attached to the board, is very difficult to control properly. My intention was that the snowboarder would move the board using his leg joints in order to change his direction and maintain balance. However, it is not easy to change the direction of the board this way. In order to reposition the board, snowboarders perform a slight jumping movement which is time-dependent and doesn't emerge as a function of the angle of the body. The snowboarder did achieve some level of control by rotating his hip and leaning forward and back, and he balanced himself this way in some parts of the terrain but not in others. It is also possible that the neural network was overfitted for the training conditions.

8. Cyclist

The cyclist was modeled as shown in Figure 5. In terms of degrees of freedom, the model is similar to the one used by [Hodgins et al. 1995], except that the crank is not modeled. The cyclist glides down an inclined path, using his potential energy to achieve movement. The cyclist's legs were not modeled either. The handlebar is attached to the fork with a rigid joint. The fork is attached to the main bicycle body and is able to rotate around its own axis. Therefore, rotating the handlebar causes the fork to rotate. Finally, the front wheel is attached to the fork and rotates with it on a steering movement. The cyclist's body attaches to the bicycle body through a rigid joint.

The arm and hand joints are the most important parts of the system. The shoulder joints are ball-and-socket joints with three degrees of freedom. The elbow joints have one rotational degree of freedom and are the only actuated joints in this model. The hand joints have two degrees of freedom. Since I wanted to try to use only the elbow joints for control, it was critical to experiment with these types of joints to achieve the correct controlability. If the hands had one more degree of freedom, essentially becoming ball joints, the arms would be able to rotate and bend down unnaturally. If, on the other hand, one degree of freedom was removed, either on the shoulders or on the hands, the character was no longer able to turn the handlebar.

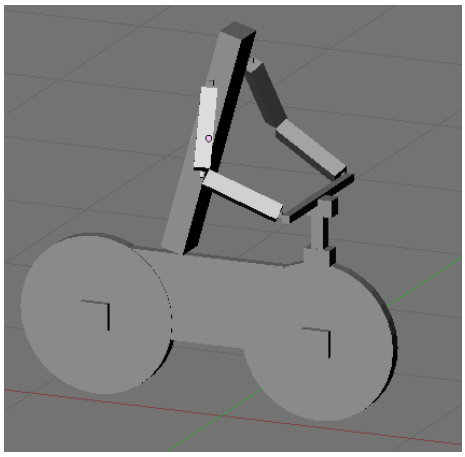


Figure 5: Cyclist

The cyclist's neural network consisted of two inputs, which are the angle the character makes with the vertical projected on the bicycle's lateral direction, and the speed of the characters. The outputs are the two desired angles for the elbow joints. Although only one would be necessary, I used two in order to make the correct opposing control emerge from the neural network. One hidden layer with four neurons was used.

Once the correct set of joints was defined, the cyclist went through his annealing process. The results were very positive, as can be seen on the accompanying video. After learning, the cyclist can ride for a good distance on almost any terrain without falling, as long as there is some initial velocity to provide the possibility of control, and as long as the direction is somewhat oriented towards the descending slope to continuously provide kinetic energy.

9. Results

A video demonstrates the results for all three characters. The cyclist is demonstrated first, then the snowboarder and

the spider. It can be downloaded from: http://www.cs.ucla.edu/~epoyart/annealing/.

The spider learned to walk, although the leg movements don't seem very natural when examined closely. It is possible that a global maximum was not achieved with this character. Its locomotion speed is also very slow. However, it is still interesting that the spider's frequency-domain controller was able to achieve movement in some way.

The snowboarder learned to glide short distances without falling. Although I think the snowboarder results were unsatisfactory, it did develop a certain level of control with the twisting of the hips, which makes it more stable than if this control was not present.

The cyclist glides long distances without falling. This character was the one that achieved the best results. The results can be tested using the interactive system. By slanting the cyclist to the left or to the right using the mouse, one can observe the resulting arm movements that control the handlebar and steer the bike. One can use the mouse to set the cyclist upright anywhere on the terrain roughly facing the right side, and give it a slight initial velocity. When that is done, it is almost certain that the cyclist will maintain balance and glide down the terrain.

10. Conclusions and Future Work

I implemented a system that allows articulated models to learn to locomote. This system has a modeling pipeline, consisting of an exporter that can be used in Blender, and a modular run-time component, allowing it to be data-driven in terms of the model used. The run-time system performs simulated annealing either on a frequency domain controller or on a neural network. It is successful in making models learn to locomote. Positive results were obtained for the spider and the cyclist, especially the latter. When in play-back mode, the run-time system is interactive, allowing the user to reposition the model to test its effectiveness under different starting conditions. When in annealing mode, the visual annealing feature allows the experimenter to have some level of insight on the process. By observing the first few hundreds of iterations (which usually doesn't take more than a few minutes), he/she can abort the process and change the parameters if it doesn't seem to be converging to good results.

Future work can be done in the modeling pipeline and the Blender exporter. The *Model* class need to be cleaned up and made more modular, since I added some functionality that doesn't belong there. This functionality, which is specific for the models used in this project, would be a better fit for the *Simulator* class.

More work can also be done in trying to develop a more controllable snowboarder model. The modeling system and its support for easy changes and experimentation will help that. The snowboarder model is too rigid on the legs, feet and board area, and better joint arrangements can be devised. This is more critical in the case of the snowboarder than the cyclist, since all of the snowboarder's weight is supported by his legs.

Finally, experiments need to be done with parallel simulated annealing, which can greatly improve simulated annealing performance in multicore machines.

References

Baraff, D. 1997. *Physically Based Modeling: Principles and Practice.* SIGGRAPH Online Course Notes.

Blender. Open source 3D content creation suite. http://www.blender.org/

Grzeszczuk, R.; Terzopoulos, D. 1995. *Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction.* Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 63–70.

Grzeszczuk, R.; Terzopoulos, D.; Hinton, G. 1998. *NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models.* Computer Graphics Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, pp. 9–20.

Hodgins, J. K.; Wooten, W. L.; Brogan, D. C.; O'Brien, J. F. 1995. *Animating Human Athletics*. Proceedings of the 22nd annual conference on Computer Graphics and Interactive Techniques, pp. 71-78.

Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. 1983. *Optimization by Simulated Annealing.* Science, New Series, Vol. 220, No. 4598, pp. 671-680.

ODE – Open Dynamics Engine. http://www.ode.org/

Poyart, E. 2010. *Interactive Articulated Rover on Mars.* http://www.cs.ucla.edu/~epoyart/rover/

Terzopoulos, D.; Tu, X.; Grzeszczuk, R. 1994. *Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world.* Artificial Life, 1(4):327–351.

Web 3D Consortium. *Open Standards for Real-Time 3D Communication.* http://www.web3d.org/x3d/