Physics-based motion control through hierarchical neuroevolution

Mart Hagenaars, Nicolas Pronost, Arjan Egges

Physics-based motion control through hierarchical neuroevolution

Mart Hagenaars
Utrecht University
m.j.p.hagenaars@uu.nl

Nicolas Pronost
Utrecht University
nicolas.pronost@uu.nl

Arjan Egges
Utrecht University
j.egges@uu.nl

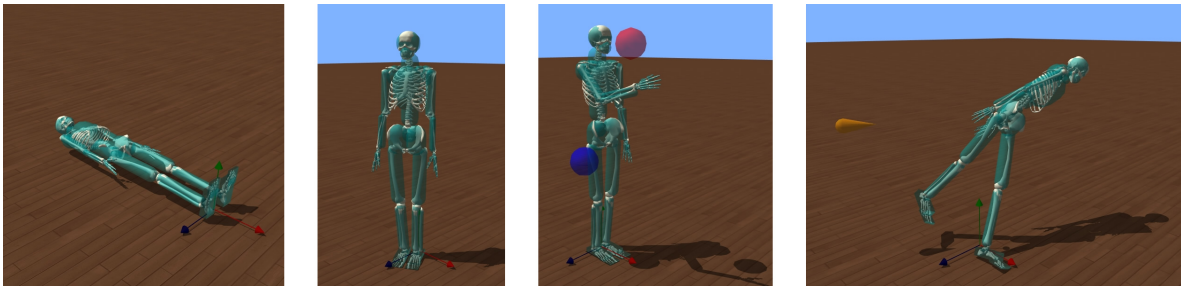

Figure 1: From left to right, behaviors of the cumulative control modules for posing, standing, and reaching. The last one shows the standing controller reacting to external perturbations.

Abstract

In this paper, we propose a hierarchical neuroevolution technique for physics-based character animation control. The artificial neural network that makes up the controller is composed of a number of interdependent control modules. As a proof-of-concept, modules for posing, standing, and reaching motions are demonstrated. We show that evolving these modules one-by-one, with each of them dependent on its predecessors, allows evolution to converge faster, and possibly deliver better and more stable results than common, i.e. non-hierarchical, controllers.

Keywords: Developmental hierarchy, Neuroevolution, Motion control, Physics-based character animation

1 Introduction

Before an infant learns how to walk, it has usually mastered several other motor skills, such as rolling over, crawling, and sitting upright. Some of these abilities are necessary in learning to walk, as they lead to the development of both the required neural pathways and physical strength. Such a hierarchy of motor skills in humans has already been identified [1]. Our approach is inspired by this concept which we apply on the neuroevolution of sensor-motor skills.

Allen and Faloutsos [2] use the NEAT [3] algorithm to optimize artificial neural networks (ANNs), similar to what we propose in this work. Their approach allows for gradual growing of the neural network, allowing for increasingly complex behavior. They do not assume any a priori knowledge of the appropriate actuation patterns, but require only the physical properties of the character model and a simple fitness function. In contrast, the controllers that we present are not dedicated to just a single task (e.g. locomotion), but rather for a hierarchical set of different behaviors that can be dynamically chosen.

The main contribution of this paper is the method that we propose to generate joint torque-based animation controllers, which consists of an ANN that is created through a process of artificial evolution. This is achieved by imposing a novel hierarchical and modular design, in which the evolution of high-level functionality is done step-by-step, as guided by multiple simple objectives, instead of by a single, complex

fitness function. The evolution process can be repeated to build ever more complex controllers, ultimately resulting in multiple control modules that form a *developmental hierarchy*. Our approach is based on the premise that it is easier to train an ANN for a new task if it already has the solutions to more basic tasks at its disposal.

2 Methods

In our approach, evolved controllers, that generate actuation patterns, are created through an off-line optimization process, so that they may be used in real-time applications afterwards. Our virtual character consists of 16 rigid bodies and 15 joints (see Figure 1). Polygon mesh data is also available to visualize the bones inside each body. The axis-aligned bounding box for each mesh is used to generate simple collision shapes; solid boxes (flat feet) or capsules.

2.1 Developmental Hierarchy

The animation controller is built from ANNs. The topology and the connection weights that are associated with the connections determine how information can be processed by the network. The NEAT method [3] is used to optimize both the weights and the topology of the ANNs for the proposed fitness functions.

The hierarchy consists of one control module per motor skill where each control module is an ANN. The first control module represents a low-level motor skill. Once the first module is fully trained for, the second module is added to the controller, representing a higher-level motor skill. During the training phase of the second module, the first module is active but is not evolved anymore. In the same way, a third module may benefit from its underlying motor skills, and so on, until the top-level module that represents the final desired behavior is completed.

Each control module is an ANN that has input nodes (for control and sensors) and output nodes (for actuation). Conceptually it makes no difference whether the output values are interpreted as joint torques, desired joint angles, or muscle activations, because the control modules are trained accordingly during evolution.

Because many control modules use the same sensory information, while producing output for largely the same actuators, a single set of input and output nodes is shared by all modules across the animation controller and they are then immutable by evolution.

2.2 Application

For the proof of concept of our method, the following target behaviors and fitness functions are used, intuitively ordered in terms of complexity.

Posing One of the most basic actions that is generally performed with virtual characters is keeping them in a desired body pose. In a physics-based environment, this means that a character should actively compensate for any perturbations from external forces, to keep a predefined rigid body pose. The fitness function of the posing control module consists of two objectives. The first one, $E_{restPose}$, is used to converge towards a particular body pose, as defined by target joint angles $\overline{\theta}_j$.

$$E_{restPose} = \sum_{j \in \mathcal{J}} \sum_{t} (\theta_{j,t} - \overline{\theta}_{j,t})^2 \qquad (1)$$

This can be done for either all simulation frames t, or for just the end state, which is the second posing objective $E_{restPoseEnd}$ (where the sum over all simulation frames is removed from the equation). The joint degrees-of-freedom (DoF) j are in a set $\mathcal{J}$ that comprises a specific number of joints, that depends on which part of the body needs to be constrained.

Standing In the standing controller, the character should keep a pose but also remain upright, with both feet on the ground, i.e. it should not fall down. In addition to the objectives from the posing controller, we use the following two objectives [4].

$$E_{COM} = (\mathbf{c} - \overline{\mathbf{c}})^2 \qquad (2)$$

where E_{COM} is the squared distance of the character's center-of-mass $\mathbf{c}$ to a target position $\overline{\mathbf{c}}$ at the desired height. This leads to the behavior of the character trying to keep its balance.

$$E_{feet} = y^2_{leftFoot} + y^2_{rightFoot} \qquad (3)$$

where E_{feet} is the sum of the squared altitudes of both feet, where minimizing leads to the feet staying on the ground, which helps in finding a balanced pose.

During evaluation of a standing individual, randomized external forces are applied at the center of the character's pelvis, to enforce learning of corrective motion to keep balance.

Reaching The reaching behavior is defined as moving the right hand to a target position, while keeping a balanced stance. After reaching the target, the hand returns to its original position. This behavioral pattern is repeated indefinitely.

The reaching module uses all of its predecessors' objectives, and adds two of its own. The main reaching error metric is the sum of two components, representing moving the right hand $\mathbf{h}$ towards the target position $\overline{\mathbf{h}}_{target}$, and moving back to the rest pose $\overline{\mathbf{h}}_{rest}$. The notation of t_f is used for those simulation frames that are spent moving the hand towards the target, and t_b for the ones moving back.

$$E_{reach} = \sum_{t_f} (\mathbf{h}_t - \overline{\mathbf{h}}_{target}) + \sum_{t_b} (\mathbf{h}_t - \overline{\mathbf{h}}_{rest}) \tag{4}$$

An error metric for control torque is added to prevent jittery motions of the reaching arm.

$$E_{torque} = \sum_{j,t} \tau_{j,t}^2 \tag{5}$$

is the sum of the control torques τ over every DoF j of the joints, for all simulation frames t.

The behavior of alternating between reaching for the target and the rest pose is governed by a finite state machine.

The fitness values are calculated using the following formula, where f_W is the fitness value, given a set W of error-weight value pairs (x, w).

$$f_W = 100 \cdot \left(\frac{\delta t}{T} \prod_{(x,w) \in W} 1 + \frac{w}{1+x} \right) \tag{6}$$

The number of simulation frames that have passed without individual failing is denoted δt, with relation to the target number of frames T.

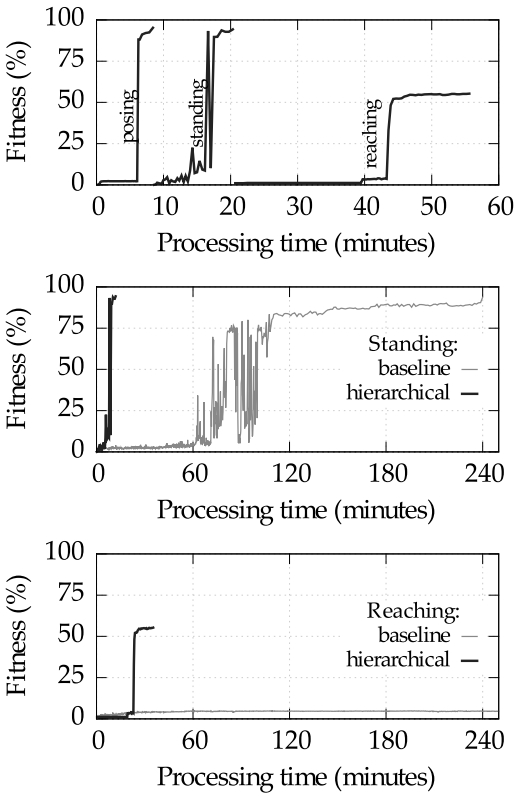

Figure 2: Evolutionary progress of the hierarchical and baseline controllers.

An individual fails, as a form of early termination, if one of its metrics exceeds a particular threshold, which depends on the control module and is fine-tuned by hand.

We interpret the controller outputs as target joint angles, which are then converted to joint torques using Proportional Derivative control. Each joint DoF has a PD controller, using the same k_p gains and torque limits as [4]. The k_v values are set to $2\sqrt{k_p}$. The resulting torque is clamped to be within $\pm 200Nm$. To limit vibration of the toes, we add passive spring-dampers (spring constant of 30 Nm / rad).

3 Results

A set of animation controllers, generated using the proposed method, is compared to a set of baseline, non-hierarchical, controllers (see Figures 2 and 3). Without exception, the baseline controllers took longer to evolve (if at all), or otherwise achieved lower levels of fitness.

Since posing is, in this case, an elementary behavior (meaning that it has no underlying

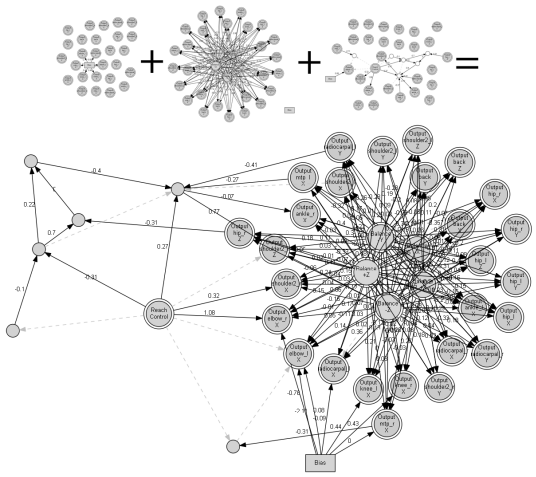

Figure 3: Posing, standing, and reaching networks are hierarchically combined.

behaviors), the corresponding hierarchical and baseline control modules are exactly the same. Therefore, no comparison is needed.

As can be seen in the complementary video, the result of both standing controllers is a balanced stance of similar quality. The baseline character adopts a slightly wider stance, which may provide better lateral balance.

As opposed to the hierarchical reaching controller, the evolution of the baseline reaching controller has to start out with a long period in which it learns to keep a balanced pose. In later generations, the reaching arm is jerked upwards, independent of whether the reaching state is active or not. This causes the character to lose balance, and fall to the ground. Even after passing a time limit of 4.5 hours, evolution does not converge to an acceptable solution.

The hierarchical modules are coming out ahead in terms of both the time and number of generations that are needed to converge to a solution. The hierarchical controller is generated in under an hour of evolution, while the baseline controllers take several hours apiece. Also, generations for the baseline controllers seem to take less time (19.9 versus 17.9 sec / generation on average for the standing controllers), but they do require many more of them.

Another interesting observation is that the baseline modules tend to evolve more connections and hidden nodes. This is probably due to the properties of complexification over time that are inherent to the NEAT algorithm.

4 Conclusion

In this work, a hierarchy of control modules is applied to the evolution of a physics-based character animation controller. We have shown that evolving these control modules one-by-one, with each of them dependent on its predecessors, will allow evolution to converge faster than when using baseline controllers. The control modules can be dynamically switched on or off, resulting in a variety of different tasks that can be performed by the same hierarchy.

Evaluating the naturalness of the results proved to be difficult. Both baseline and hierarchical animation controllers generate motion that is showing some unrealistic characteristics. In the future, we plan to test our approach with larger hierarchies, thus capable of generating a larger variety of motion where behaviors are chained to perform more complex actions. For example, we want to design controllers for locomotion and combination of locomotion with other tasks such as reaching.

Acknowledgments

This research is funded by the European Community Seventh Framework Program, under the TARDIS project (FP7-ICT-2011-7).

References

[1] S. Uithol, I. van Rooij, H. Bekkering, and P. Haselager. Hierarchies in action and motor control. *Journal of Cognitive Neuroscience*, 24(5):1077–1086, 2012.

[2] B. Allen and P. Faloutsos. Evolved controllers for simulated locomotion. *Motion in Games*, pages 219–230, 2009.

[3] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[4] M. Al Borno, M. de Lasa, and A. Hertzmann. Trajectory optimization for full-body movements with complex contacts. *Visualization and Computer Graphics, IEEE Transactions on*, 2012.