CGI2012 manuscript No.
(will be inserted by the editor)

Generating Real-time Muscle Activations for Skeletal Hand Motion: An Evolutionary Approach

Arun Somasekharan · Hammadi Nait-Charif · Jian Jun Zhang

Abstract This paper presents a system that generates muscle activations from captured motion by borrowing principles from biomechanics and neural control. A physics engine compliant muscle system is developed using the human hand as an example. A machine learning approach using evolutionary neural networks is adopted for creating the muscle control system and dynamical simulation is performed using a real-time physics engine that is used in present day games. The system was trained for a single finger and then tested on the five fingers of the hand. The simulation results show that the system produced a close mimic of the motion-captured hand animation. The system also produced a believable animation from motion data not present in the training set.

Keywords Artificial Neural Networks · Genetic Algorithms · Physics Engines · Games · Animation

1 Introduction

Physics-based animation of characters using anatomically similar muscles is a complex problem involving both activation dynamics and force calculations, making it computationally expensive [1]. Coordinating complex motion of articulate characters automatically at a physical level requires some form of controller architecture. Proportional Derivative (PD) controllers have gained a lot of popularity in the field of physically-based animation, due to their simple linear formulation and feedback nature [2]. PD controllers have been used in animating a Luxo Jr lamp character [3], grasp animation of hands [4]. Through a post-stabilisation process, they have also been used as line segment muscle actuators in biomechanical simulations [5]. Evolving controllers for coordinated motion in physics-based characters have been researched in the field of Artificial Life. Sims in [6] evolved both morphology and the neural controller of virtual creatures. [7] uses neural networks to learn muscle control of the human neck. Neural networks have also been used in learning muscle activation patterns from Electromyographic (EMG) recordings for biomechanical simulations of the human body [8]. But a majority of the implemented algorithms exist in customised simulation environments that often compromises their compatibility with generic physics simulators often found in games [1] [5]. Due to these reasons, most of these methods are unsuitable for physically animating game characters. Thus, a motor control system capable of generating real-time muscle activations can be invaluable in creating physically-based animation of game characters. Such a system would be able to produce localised damages on game characters by de-activating the muscles in the proximity of the damaged area. This type of effect is impossible with current PD controllers. This paper proposes a machine learning approach that defines a functional mapping between a motion envelope and activations of musculature to produce the same motion. The muscle activation space is explored indirectly using Genetic Algorithms (GA). The contributions of this paper towards the creation of such a controller are:

– An artificial neural network (ANN) that predicts muscle activations which correspond to the direction of movement in motion data.

Arun Somasekharan
asomasekharan@bournemouth.ac.uk

Hammadi Nait-Charif
hncharif@bournemouth.ac.uk

Jian Jun Zhang
jzhang@bournemouth.ac.uk

– A force-scaling method that allows for timing control in physical animation.
– A simplified linear piece-wise line segment based muscle model that is optimized for a generic commercial quality real-time game physics engine. From a real-time perspective, this simplified model is very important.

The remainder of the paper is structured as follows: section 2 describes the controller system and muscle actuator system, section 3 presents the simulation results and section 4 concludes this paper.

2 Real-time Muscle Activation System

A high level muscle system is implemented that conforms to generic physics engine standards. The motor control system produces coordinated muscle activations on a per frame basis. It primarily consists of a monolithic feed-forward artificial neural network consisting of an input layer, one hidden layer and an output layer. Input to the network are pose vectors calculated from motion capture data. Genetic evolution is performed to select the best network from a stochastically generated population of networks. Fig. 1 shows the component-wise architecture and data flow.

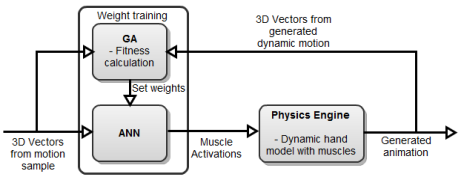

Fig. 1 The architecture of the system.

2.1 Thin Muscle Structure and Force Scaling

The motion of the rigid skeleton is governed by the contact interactions of the tendons with the underlying bones [1]. In order to simulate that, special geometry place holders, termed as *muscle locators*, are used to specify points of contact of muscular tendons with the bones. The locators also define the path of the tendons. Using vector decomposition a vector $\overline{V}$ is written as,

$$\overline{V} = \widehat{n} + \widehat{t} \tag{1}$$

These normal, $\widehat{n}$, and tangential, $\widehat{t}$, components are essential to model the contact forces prevalent at each muscle locator along the thin structure, on the rigid body linkage. Contrary to existing controllers [1] [5], this implementation accepts the force magnitude as a user-specified quantity, and the direction of the force vector is dependent on the layout of the thin muscle structure on the rigid body linkage. By making force magnitude an input quantity, control is not completely abstracted from the animator. The force associated with a thin muscle structure is modulated using an activation level a (where $0 \leq a \leq 1$), thereby producing different scaled forces per thin muscle structure. Thus the magnitude of the force on the rigid linkage is distributed among the involved muscle structures.

Using Eq.1, we can represent force scaling for a single thin muscle structure as,

$$\overline{F}_s = a\left|\overline{F}_m\right| \sum_{i=1}^{k} (\widehat{n}_i + \widehat{t}_i) \tag{2}$$

where $\overline{F}_s$ is the scaled force vector, a is the activation level for the thin muscle, $\overline{F}_m$ is the user input force magnitude, k is the number of muscle locators for the thin muscle structure, $\widehat{n}_i$ and $\widehat{t}_i$, the normal and tangential components of the contact vectors at each muscle locator on the thin muscle structure.

2.2 Artificial Neural Network

Muscle activation dynamics can be considered as a time series problem with dependency on previous states of the motion trajectory of the skeletal linkage. The neural networks used in *time series prediction* normally have standard feed-forward architecture [9]. The network accepts an N-tuple set as inputs that describe the sequence d steps back in time. This method is often termed as a *sliding window technique* [9]. So for every t+1 step, the network outputs the activations that allow the muscle to contract so that the dynamic linkages mimic the orientation of the input skeletal set. Mathematically, a predictor function can be defined as,

$$x(t+d) = f(x(t), x(t-1), ..., x(t-N+1)) \tag{3}$$

where f is a real valued function that calculates the value of the observed system at a future time $t + d$. Based on the structure of the human hand, there are two ways of creating the neural network controller. They are:

– Single neural network controlling all finger muscles as a group
– Individual neural networks for each finger.

In this paper, the second option was preferred due to the duplicative nature of the tendon layout on each finger. A fully connected feed forward monolithic neural network was created for the forefinger of the hand. The motion samples used in the training are captured hand

motions stored in the Autodesk FBX format embedded with hierarchical transformations for joints/nodes. The FBX data is converted to a simple 3D vector data format for each frame and stored as a text file (see Fig. 1). With 3 components per vector and 3 vectors per finger (for each phalange considered in the training), there are 9 inputs. Two previous states $t-1$ and $t-2$ are also used to predict the activations at time t. Thus the total number of inputs becomes 27. The outputs of the network are muscle activations. 8 muscles are used for each finger which directly dictates the number of output neurons in the ANN.

2.3 Genetic Evolution

Artificial neural networks that use evolutionary algorithms for training, architecture design etc are termed as *evolutionary neural networks* (ENN) [10]. In the ENN used in this research, the architecture is fixed and synaptic weights are evolved using the GA. The chromosomes undergoing evolution encodes the connection weights of the ANN and the evaluation is performed through suitable objective functions that rate the motion produced by each network in the population. A primary objective function is used in conjunction with a penalty function. They are:

2.3.1 Pose-based Objective Function

A simplified vector chain can be extracted from the 3D skeletal key frame. Since the dynamic model of the hand (or any articulate body) is segmented with joints that constrain the rigid bodies (phalanges), the result is a vector chain containing a set of non-arbitrary vectors. The orientations of the resulting vector chain is used for pose identification of the articulated rigid bodies.

By comparing the angles derived from the directional cosines of two sets of vector fields, it is possible to estimate the similarity in two poses. If $\overline{V_1}$ and $\overline{V_2}$ are two vectors, with directional cosine angles, $\theta_1, \omega_1, \phi_1$ and $\theta_2, \omega_2, \phi_2$ respectively, then,

$$(\theta_2, \omega_2, \phi_2) - (\theta_1, \omega_1, \phi_1) = (\varepsilon_\theta, \varepsilon_\omega, \varepsilon_\phi) \tag{4}$$

where $\varepsilon_\theta, \varepsilon_\omega, \varepsilon_\phi$ are the respective error differentials.

The representative measure of ε_θ, ε_ω, ε_ϕ termed as ε_{prime}, is defined by,

$$\varepsilon_{prime} = \max(\varepsilon_\theta, \varepsilon_\omega, \varepsilon_\phi) \tag{5}$$

The Root Mean Square (RMS), is used to calculate a representative measure for all the ε_{prime} of the n vector sets in the source and target poses.

$$\varepsilon_{primeRMS} = \sqrt{\frac{\sum_{i=1}^{n} \left(\varepsilon_{prime_i}\right)^2}{n}} \tag{6}$$

A threshold δ, which is user specified and having a very small value, is used as a comparison target for $\varepsilon_{primeRMS}$. Thus, a multi-dimensional comparison problem is reduced to a single variable comparison. So it is safe to assume that a target and a source pose are the same if $\varepsilon_{primeRMS} \leq \delta$.

2.3.2 Penalty Function

Initial test runs used only the pose based objective function for calculating the fitness of each network. But it was observed that some of the generated networks created continuous variation in activation than other networks. Constant activation causes the rigid bodies to reach a constraint maximum and maintain that pose for the duration of simulation. In order to properly isolate neural networks capable of creating variety in generated motion, a *penalty* term is introduced into the primary objective function, which penalises the fitness of the network depending on the motion generated during the training simulation. The penalty is calculated by comparing subsequent frames in all the stored poses of the generated motion by the network.

$$\xi_{accum} = \sum_{f=0}^{N} (P_{f+d} - P_f) \tag{7}$$

where P_f is the current pose and $d = 1, 2, 3....N$ is the frame offset. The sequence of poses is examined by measuring the amount of deviation between poses. This is indicative of the amount of movement generated by the network. Thus, subtracting Eq. 7 from Eq. 6,

$$\varepsilon = \varepsilon_{primeRMS} - \xi_{accum} \tag{8}$$

Thus the final fitness f is $f = \frac{1}{\varepsilon}$. The inclusion of the penalty term produces negative fitness with higher fitness closer to 0.

3 Simulation Results

The real-time physics engine, NVIDIA's PhysX was used for the simulations. The development platform was a Windows 7 laptop with an Intel i3 2.4 GHz Processor and a system memory of 4 Gb equipped with an NVIDIA GeForce GT 415M graphics card with 1Gb of video memory.

The system works completely in real-time producing frame rates of 45 frames per second (fps) on the above given configuration. Muscle visualization produces a drastic drop in frame rate (11 fps), but in its absence simulation runs very smoothly. Even though there were 6 motion samples used, only one sample was used for training since the sample had sufficient variation for

generalisation. The generalised network was replicated for the remaining 4 fingers to produce the complete hand animation.

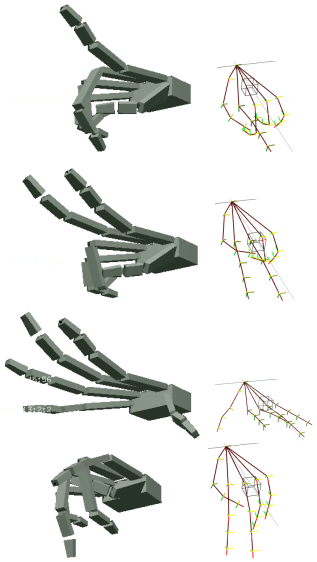

Fig. 2 A few key frames captured from the generated animation. Dynamic hand model on the left and motion capture sample model on the right.

The population size used was 50. Since deciding on the number of hidden neurons is more of a heuristic than clearly defined through a formula, trial and error was relied on heavily. Earlier runs using 10-15 hidden neuron failed to provide any useful results. Successful convergence occurred with hidden neuron number set at 30. The GA was able to produce a good solution in 32 generations. Standard GA operators were used in the beginning, without any noticeable difference. Simulations were run for as long as 100-120 generations without achieving convergence. Previous research, like in [11] and [12], suggests that cross-over lacks efficiency in real-valued evolutionary neural network problems involving dynamic learning. So mutation along with an *elitist* selection, was preferred over crossover thus preventing high divergence or *genetic drift* in the individuals of the population. A dynamic hand model was constructed using reference scaled rigid blocks and joint constraints. Fig. 2 shows key frames from the generated animation using an unknown motion sample as input. The laterality in both the dynamic hand model and the motion capture model used in training, are different - the dynamic hand model is a right hand while the motion capture model is a left hand. The trained neural network also generalises to unknown motion samples. From a production point of view, the system would be useful in converting a kinematic animation (which was either key-framed or motion captured) into a physics-based dynamic animation. Since the input to the ANN are vectors, a traditional IK driven skeleton can be used to activate the muscles. An in-game application would be situation induced damage on the characters where de-activating the muscles corresponding to a damaged body area would result in motion that is adaptively handicapped. This type of effect is not possible with conventional PD controller driven physics characters.

4 Conclusion

We have presented a system that successfully maps muscle activations to motion captured skeletal animation using a hybrid machine learning method using evolutionary neural networks. The system was created with due consideration to hand animation in a real-time physical environment that is commonly found in games and other real-time media.

References

1. S. Sueda, A. Kaufman, D.K. Pai, Musculotendon simulation for hand animation, ACM Transaction Graphics. Vol. 27, 3, pp. 1-8 (2008)
2. M. van de Panne , R. Kim , E. Flume, Virtual Wind-up Toys for Animation, Proc. of Graphics Interface '94, pp. 208-215 (1994)
3. L. Gritz , J. K. Hahn, Genetic Programming for Articulated Figure Motion, J. of Visualization and Computer Animation, Vol. 6, pp. 129-142 (1995)
4. N. S. Pollard, V. B. Zordan, Physically based grasping control from example, 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation, (2005)
5. R. Weinstein, E. Guendelman, R. Fedkiw, Impulse-based control of joints and muscles, IEEE Trans. on Visualization and Computer Graphics, Vol. 14, 1, pp. 37-46 (2008)
6. K. Sims, Evolving Virtual Creatures, Proc. of the 21st annual conf. on Computer Graphics and Interactive Techniques, pp. 15-22 (1994)
7. S.H. Lee, D. Terzopoulos, Heads up!: biomechanical modeling and neuromuscular control of the neck, ACM Transactions Graphics, Vol. 25, 3, pp. 11881198 (2006)
8. S.D. Prentice, A.E. Patla, D.A. Stacey, Simple artificial neural network models can generate basic muscle activity patterns for human locomotion at different speeds, Experimental Brain Research, Vol. 123 pp. 474-480 (1998)
9. R. J. Frank, N. Davey, S. P. Hunt, Time series prediction and neural networks, J. of Intelligent and Robotic Systems, Vol. 31, 1, pp. 91-103 (2001)
10. X. Yao, Evolving Artificial Neural Networks, Proc. of the IEEE, Vol. 87, 9, pp. 1423-1447 (1999)
11. T. Reil, P. Husbands, Evolution of central pattern generators for bipedal walking in a real-time physics environment, IEEE Trans. on Evolutionary Computation, Vol. 6, 2, pp. 159-168 (2002)
12. R.A. Watson, G.S. Hornby, J.B. Pollack, Modeling Building Block Interdependency, Proc. of Parallel Problem Solving from Nature V (PPSN V), pp. 97-106 (1998)