EG UK Theory and Practice of Computer Graphics (2006)
M. McDerby, L. Lever (Editors)

Evolving Body Kinematics for Virtual Characters

C. Gatzoulis, W. Tang, W. J. Stoddart

School of Computing, University of Teesside, United Kingdom
c.gatzoulis@tees.ac.uk

Abstract

Physically-based character animation systems often require complex knowledge of the underlying equations of motion. Hence, producing physically-realistic animations can be time consuming with these systems. In this paper, we present an approach that automatically searches for kinematics solutions for virtual characters. Characters learn their locomotion by evolving body kinematics. We designed two different control architectures for the character's learning process with predefined motion data sets and a feedback system. The first system is based on a layer of genetic algorithms (GA) and the second is based on a Reinforcement Learning (RL) approach. Animation systems based on these control architectures require little knowledge of the physics equations of motions, but can generate physically-feasible motions in real-time through observations of available motion data sets, such as previous animations or motion capture data. This animation approach allows animators to construct easily realistic body kinematics motion for computer game characters. Embedded with simulated musculature of human body, the system also has applications in sports and physiotherapy for motion visualization. The test data also demonstrates the advantages and drawbacks of the two types of control methods.

1. Introduction

Physically convincing body motions for virtual characters are desirable in 3D computer animations and computer games. Many research investigations involved the use of physically-based modelling and simulation to produce realistic animations for virtual characters [ALP04, ZMCF05]. As physically-based motion animations are computational expensive and real-time solutions do not always succeed, motion capture data sets are commonly used in animation productions. With predefined key frame animations and fixed sets of motion data, virtual characters lack adaptive abilities for dealing with different situations in a virtual environment. The attempts made to incorporate a layer of physical properties to motion data have the inherent difficulty as in physically-based animations [ZV03, DYP03]. To overcome the obstacles inherent with physical simulations, probability and statistical method was applied to learn motion patterns from various captured motion data sets [BH00].

Recent novel approaches to create physically-feasible and adaptive character animations use artificial intelligence techniques to synthesize the motion controller embedded in the character's skeleton structures [FV93, WT02, E05]. One approach used evolution algorithms such as genetic algorithms [WT02, E05, GT95] and reinforcement learning [IBDB01] to generate individual motion behaviours for virtual characters. The choice of the algorithm embedded in the motion controller for the action-selection mechanism is crucial to the results of the animation. It is important that the chosen algorithm is able to generate realistic motions efficiently. Especially for real-time animations, the design of the control architecture plays a major role in gaining the performance of the algorithm.

In this paper, we present two types of motion controller for generating virtual characters that can learn and adopt body kinematics motions in the similar way that humans learn how to perform movements, such as for walking based on observation, trial and error. Furthermore, characters need to have the ability to generate motions that are unique and dependent on their experience. As in reality, our animation system embedded with the controllers generates motions for characters that have individual characteristics that distinguish each character from others. By incorporating such a type of learning behavior, the generated virtual characters are adaptive with evolving motion dynamics without the embedded underlying equations of motion.

Section 2 of the paper relates our work to important previous work. In section 3, the construction of the character animation structure is introduced, which supports the underlying control architecture for the adaptive

animation. Section 4 and 5 describe details of two types of AI controllers, namely, genetic algorithm controller and Q-learning controller. Experimental results on the two controllers are shown in section 6 and section 7 concludes our work and highlights directions of the future work.

2. Related Work

Synthesizing adaptive motions in the context of learning and evolution is based on the construction of a controller. A controller makes decisions based upon sensory information received or available and evaluates the outcome solution through a feedback scheme. Popular artificial intelligence methodologies are genetic algorithms [HOL75] and reinforcement learning algorithm such as Q-learning [SB98].

Genetic algorithms were used to model stimulus-response pairs for the locomotion of 2D stick figures [NM93] and for improving sensor-actuator networks for stick figure motion [FV93]. An effective feedback control system was able to create virtual creatures with their body structures evolving in order to compete with each other in a virtual environment [SIM94]. The control system used GA to simulate morphologies and the neural systems for controlling muscle forces of the virtual creatures. A GA based approach was presented for virtual characters to learn physically based motion behaviors as an evolution process. The skill of the character to complete a predefined task can be developed and evolved through the experiences of performing the task [WT02]. As a reinforcement learning method, Q-learning algorithm was applied to generate action sequences for animated characters [SF04]. Based on an animation engine, a behavior engine used a script generator for selecting motions from a database and a characterizer for generating characterized motion by using evolutionary computation [NKL*04]. Using genetic algorithms, dancing motions for arbitrary songs with dance beats were automatically synthesized [ABB05]. A dynamic motion synthesis system was proposed to create interactive 3D characters [BPW93].

GA and Q-learning have their own merit and drawback, thus when designing an efficient controller for adaptive animation, one must consider the merit and drawback of GA and Q-learning algorithms. As a controller designed by GA is a closed set feedback system that has limited external stimulus/response, the principle advantage is the efficiency of the simulation. On the other hand, Q-learning works by constructing a look-up table that is updated by evaluating the task performance of the character. Compared with GA, in the Q-learning algorithm external stimuli can be easily incorporated into the feedback system to generate environmental aware characters. In the following sections, we present the control architecture and the implementation of our animation system. We constructed two types of controller, using GA and Q-learning algorithms respectively. In order to gain insights into the efficiency of the controllers and to produce robust adaptive motion, tests are conducted with body kinematics for virtual characters.

3. Character Animation Structure

The animation structure needs to support the underlying control architecture. A character animation structure is created using a set of kinematic chains starting from the root to extremities. Motions of a character are stored as a series of transformations taking place in an animation sequence. An articulated body skeleton is represented by a total of 17 joints as shown in Figure 1. The joint j_1 is the base joint and the limbs are animated with rotation transformations relative to this base joint. The base joint requires a translation and a rotation. The transformations of the limbs are stored using a reference frame attached to the root of the corresponding kinematic chain (shoulder or hip).

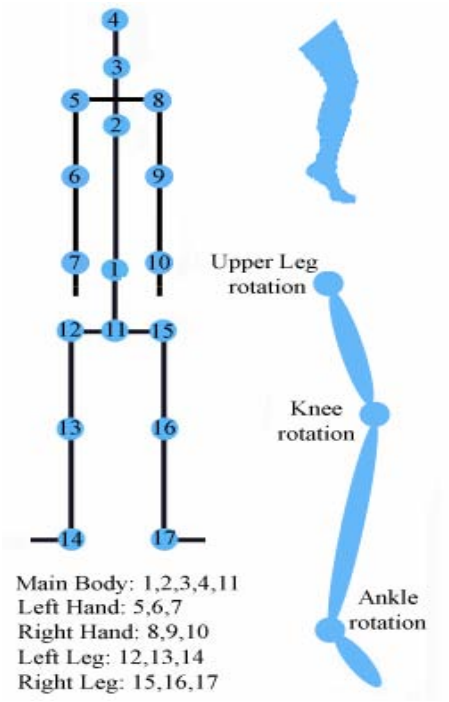


Figure 1: Animation skeleton.

Each transformation is described with three values, corresponding to X, Y and Z axes respectively. The movement of the character is described by a set of animation frames. The controllers described in the following section utilize the skeleton structure by attaching a sensory node to each of the joints. Hence, the resulting adaptive animation is generated by a connected sensory network with feedback evaluation optimized motions.

4. Designing the GA Controller

As shown in Figure 2, an evolution control module uses the learning algorithm to generate data for the motion parameters. The initial parameters are used by the motion generation module to calculate the values of transformations for each frame. The motion of each animation frame is appraised by the evaluation module based on the target task. A threshold is defined to estimate the error in the motion parameters and a feedback message is generated for the evolution control unit. The evaluation process is iterated until an optimal motion is found.

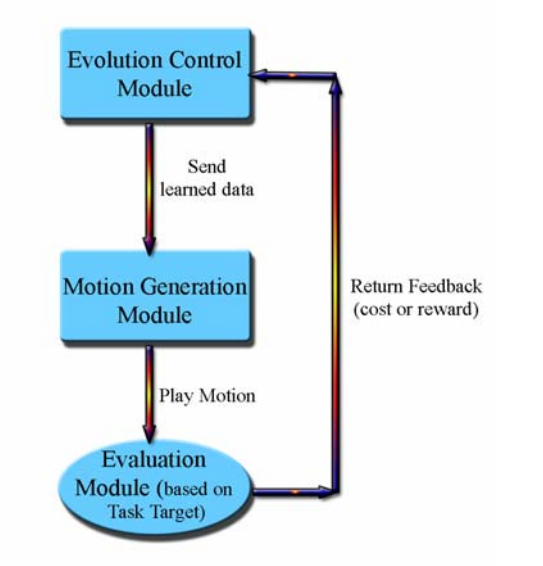


Figure 2: Control Structure of system flow

We design the joints of the skeleton as sensory nodes, with bones between these joints linked to form a sensory network. In the GA controller, the network is considered as a chromosome and the genes of the chromosome hold the values of the transformations of the joints. Each separate chromosome constitutes a motion frame.

The initial set of chromosomes is created randomly and using the natural selection scheme of the GA algorithm, only the fittest ones are kept after a sorting algorithm. A pairing strategy sets the pairs of chromosomes to be the parents that will generate offspring chromosomes for the proceeding iterations with a mating strategy. To allow the algorithm to converge more quickly to an efficient solution, a level of exploration is achieved by mutating the genes of offspring chromosomes.

The new chromosomes need to be assigned a cost value and the total selection is then sorted. The feedback system evaluates the calculated total cost of all the used chromosomes. The fittest chromosome is passed to the motion generation module. The interaction process stops when the value of the cost is below a threshold and the motion held by the current chromosome is considered acceptable. Otherwise the iteration process continues. This feedback scheme is illustrated in Figure 3.

Different strategies in a genetic algorithm can be used [HH98]. When creating the initial chromosomes, it is helpful to generate more chromosomes than required and keep the fittest ones after the initial selection. We keep 50% percentage from the initial population. The cost function evaluates the learned data as in Equation 1.

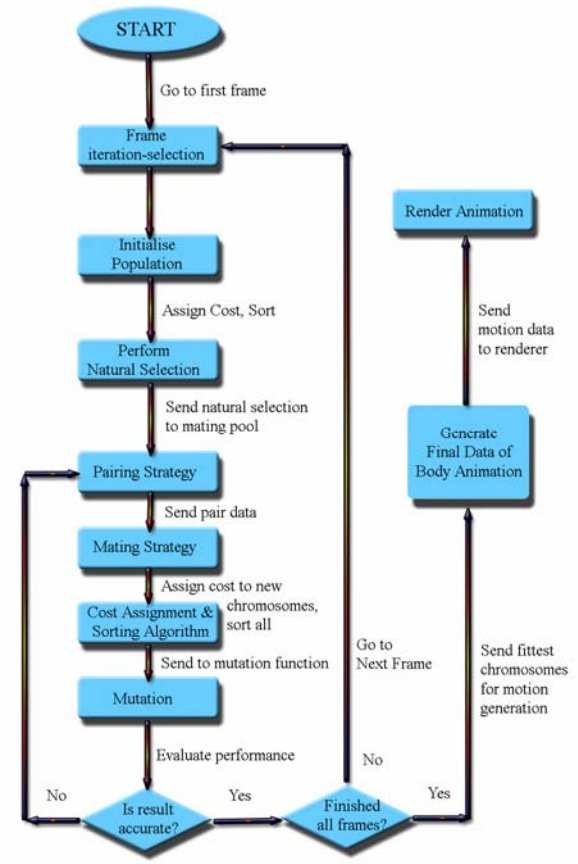


Figure 3: Design of GA process

C_l is the learnt data, C_o is the original data and N_p is the total number of genes in a chromosome. Cost is also dependent on other outputs such as visual results of the motion and resulting body states if a physically based body simulation exists.

$$Cost = \sum_{i=0}^{Np} | C_l - C_o | \quad \textbf{(1)}$$

Rank weighted paring is one of the strategies that can be used to assign probabilities to the chromosomes by taking into account their cost. As shown in Table 1, each chromosome has five genes and the chromosome number 1 has the highest weight. The rank weight P_n is calculated as follows:

$$Pn = \frac{Ng - n + 1}{\sum_{n=1}^{N_g} n} = \frac{5 - n}{10} \qquad (2)$$

Ng is the total number of chromosomes that have the weight above threshold and n is the total number of chromosomes in the system.

Table 1. Rank Weighted GA pairing.

n	Chromosome	Pn	Cum. Pn
1	1.5 2.1 0.4 4.2 2.1	0.4	0.4
2	1.0 2.6 5.4 3.1 2.6	0.3	0.7
3	1.2 1.1 3.2 2.4 3.8	0.2	0.9
4	1.3 3.5 2.5 0.7 5.7	0.1	1.0

For the *crossover* stage of the GA algorithm, points in the chromosomes are selected to swap the left and right adjusted parameters. In the following example, the left index is three and the right is four. The genes are exchanged and the offspring chromosomes are generated.

Parent Chromosomes

Parent1 = { p11, p12, **p13**, **p14**, p15,….. , p1N }
Parent2 = { p21, p22, **p23**, **p24**, p25,….. , p2N }

Offspring Chromosomes

Offspring1 = {p11, p12, **p23**, **p24**, p15,….. , p1N }
Offspring1 = {p21, p22, **p13**, **p14**, p25,….. , p2N }

A *Blending* method sets values at genes by taking the parent's related genes as in Equation 3.

$$Offsp[i] = Fath[i] * b + Moth[i] * (1 - b) \qquad (3)$$

where b is the blending rate.

In our GA controller, the mutation rate is set as 0.05. Although mutation can frequently generate less valuable chromosomes it is the way for the algorithm to explore the state space. This is important especially when using small populations due to real-time constraints.

The original animation of a walking sequence is created in Maya for the lower part of an articulated virtual character. The skeleton of the virtual character comprises of six joints: waist, pelvis, left and right hips and knees. For each of the joints there are a number of transformations on the three world axes X, Y and Z. The walking sequence consists of 25 animation frames. Therefore with a sum of 24 transformations we can represent the walking motion data in frames. Specifically for a forward walk, we can decrease this number by creating a local world system and allowing a translation of the waist along two axes i.e. the z and y axes, and one rotation on the x axis for each of the hips and knees. This adds up to a total of six transformations.

Each chromosome is designed with genes using per frame values of all the transformations so that a chromosome has 6 genes as shown in Figure 4. The total number of families of chromosomes is 25 representing the total number of frames. We create a layer of 25 consecutively operating chromosome clusters. Each chromosome represents a value that adds up to the positioning that resulted from the previously accessed cluster.

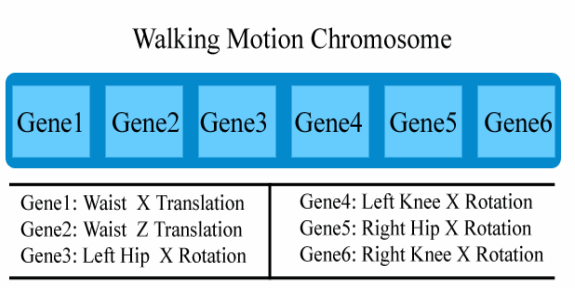


Figure 4: Walking motion chromosome

The initial population is generated by allowing the values to come from a continuous space of a constant length. The length of the space is set to the maximum increment that occurs in the original motion data set. A relatively large initial population is created. Natural selection is set at the level of 50% in order to cause no influence of unexpected selections to the results of the tests. Thresholding is avoided, as it is vague to define a function that can modify the threshold value without affecting the results of the simulation.

Figure 5 illustrates a sequence of adaptive process of the virtual character generated by our GA controller for the described walking motion. The skeleton on the right has the original motion data animated in Maya and the one on the left learns the motion through the GA controller. The number of feedback iterations for the images in Figure 5 is 1, 3, 5 and 15 respectively. As can be seen, the adaptive

character performs close to the original motion data after 15 iterations.

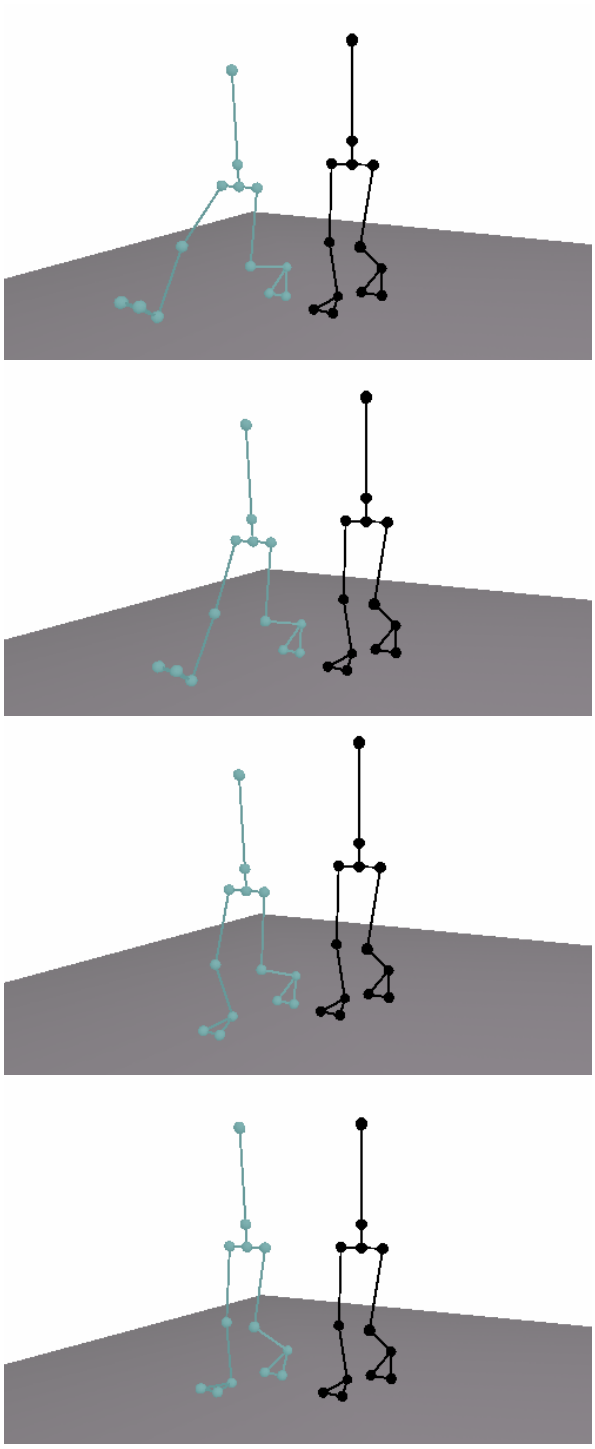


Figure 5: Results of GA evolution in walking

Tables 2 and 3 present the test results of using different pairing and mating combinations. N is the initial population, MI is the maximum allowed iterations and AC is the threshold cost value for a chromosome to be considered good. The values displayed are the average value of 25 trials for each test to ensure a stable outcome. With a large number of populations, the best result generated by cost weighted function using the blending strategy is highlighted in Table 3 as bold values. In comparison, with smaller number of chromosomes a different pairing strategy is needed to achieve optimal solution.

Table 2. GA results for learning to walk.
N=320, MI = 100, AC = 1.5

Mating / Pairing	Crossover		Uniform		Blending	
	Steps	Cost	Steps	Cost	Steps	Cost
Top - Bottom	76.55	1.83	68.01	1.75	**25.61**	1.40
Random	74.10	1.75	56.37	1.58	28.03	**1.33**
Rank Weighted	87.70	3.17	75.09	2.54	39.38	1.52
Cost Weighed	76.06	2.00	78.02	2.16	42.70	1.61
Tournament	74.23	1.86	58.95	1.74	32.72	1.48

Table 3. GA results for learning to walk.
N=800, MI = 20, AC = 1.5

Mating / Pairing	Crossover		Uniform		Blending	
	Steps	Cost	Steps	Cost	Steps	Cost
Top - Bottom	19.01	2.06	18.70	2.01	13.30	1.42
Random	19.35	2.05	17.60	1.61	15.58	1.65
Rank Weighted	19.68	2.97	17.87	1.83	14.41	1.61
Cost Weighed	18.13	1.79	16.86	1.76	**12.61**	**1.41**
Tournament	18.63	1.90	16.16	1.95	13.73	1.50

5. Designing a Q-Learning Controller

MIT [IBDB01] proposed the use of Q-Learning (QL) for adaptive synthetic characters. QL is an off-policy control Temporal Difference algorithm that describes a learnt data set in a look-up table that is updated by exploration. Drawn from our previous experimental results [TGW04], Q-Learning is chosen for our RL controller for character motion synthesis.

The major issue in the design of the QL system is how to represent a state-action space. For the state representation, each frame is assigned as a different state. Therefore, the state transition is deterministic and each state results in the state for the next frame. A set of discrete actions is defined representing the modifications in the transformation of the joints. To achieve accurate results, a real number in the space [0,1] is added to the value in order

to mimic the original data set. In this way we are able to use a small constant look-up table that helps the optimized performance in real time.

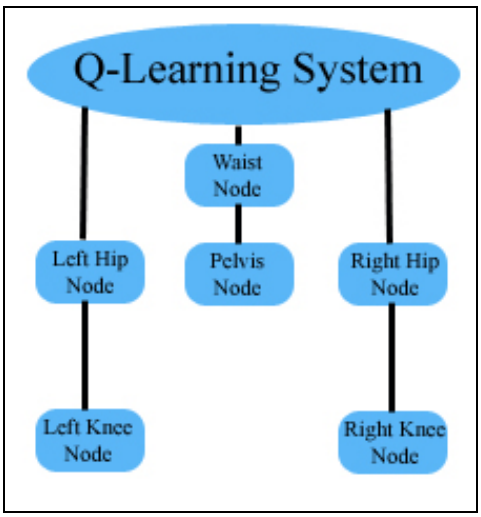


Figure 6: Layers of the Q-Learning System

The QL comprises of a layer of QL nodes each corresponding to the joint transformation as shown in Figure 6. For the walking example there are 6 QL look-up tables forming the evolving action-selection of the body motion for the character. The learned data in these nodes represents the motion solution for the virtual character.

For each of the QL nodes, the algorithm begins at state 0 and performs exploration and exploitation. When selecting an action, the related changes occur to the transformation values and the next state (frame) is visited. Each state holds a specific vector of values that are the transformation parameters. The Q-values are updated using Equation 4:

$$Q_{t+1}(S_i, a) = Q_t(S_i, a) + \alpha * \delta * E(S_i, a) \quad \textbf{(3)}$$

Here, **α** is the learning rate, **δ** is the Temporal Difference (TD) error and **E(Si, a)** is the eligibility traces value of the corresponding state-action pair. The TD error occurs when evaluating the condition of the resulting state and checking if the action was beneficial for the system or not. It is calculated as follows:

$$\delta = r + \gamma \cdot Q(S_{i+1}, a^*) - Q(S_i, a) \quad \textbf{(5)}$$

In (5) **r** is the reward, **γ** is the discount factor, $\mathbf{S_{i+1}}$ is the new state and **a*** the action chosen by the greedy policy.

In the animation system, the number of states is 25 that being equal to the number of frames. For the translations we use a set of 11 actions presenting the values of translation on the axis. For the rotations we use a set of 40 actions. Figure 7 shows the final learned solution for the adaptive character on the left compared with the motion of the original designed character on the right.

The parameters of the QL algorithm can change the performance. Based on results of our previous research we set the values of the learning rate **α** to 0.1 and the discount factor **γ** to 0.9. After 20 iterations with 20 steps per iteration, the algorithm converges to learn the walking cycle but with less accurate values for some transformations. We gradually increased amount of steps and when equal to 40, the learning character performs with much high accuracy at all attempted trials.

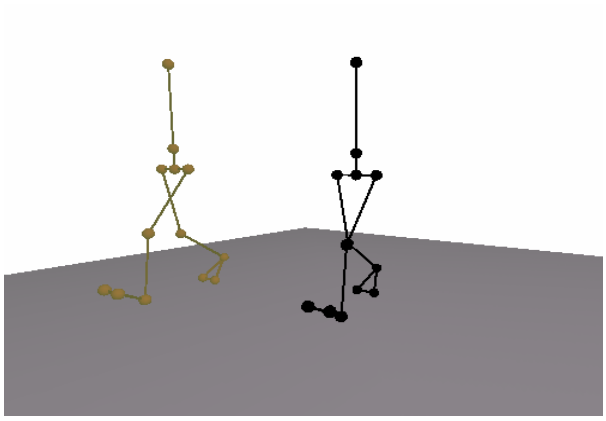


Figure 7: Results of QL evolution in walking

6. Experimental Results

In order to compare the two algorithms we measured the required time for the two different systems to learn the same animation data set.

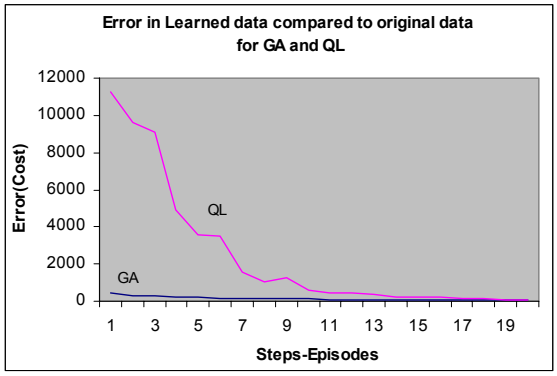
In Table 4 we compare the elapsed real-time of Q-Learning and GA controllers. For the genetic algorithms the results in the previous section are used for the choice of pairing and mating methods used for the GA controller. As can be seen, the QL controller performs within an equal amount of time as the GA with a population N=500.

Table 4. Elapsed time for the two algorithms.

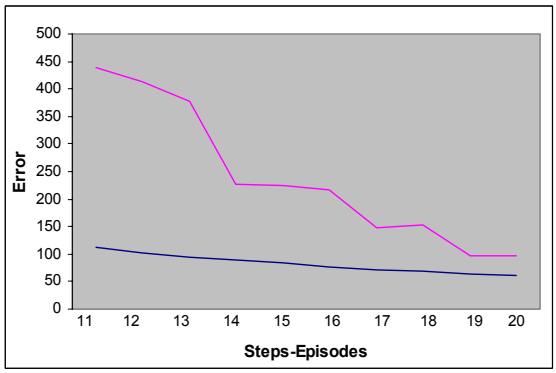
Method	Parameters	Elapsed Real-Time (ms)
QL	20 iterations, 40steps	3620
GA	N=500	3520

The rate of learning of the two methods can be seen in Figure 8. In the flowchart we illustrate the error of the algorithms per iteration step. The error is defined as the numeric distance of the learned data compared with the original data. As shown in Figure 8.a, the GA system performs with more accuracy from the early steps of the simulation. QL starts with a big error and although it improves its performance through elapsed iterations, the accumulated error at the end of the 20 steps is 50% higher than that of the GA system. The values of the error on the 20th step are 62 for GA and 96 for QL.

Using the presented methodologies, we can generate walking movements that are based on the reference motion, but each one holds individual characteristics in movement, same as these that humans can adapt when learning to walk.



8.a



8.b

Figure 8: Performance comparison of GA and QL controllers

7. Conclusion

We presented two types of control architecture for generating physically-feasible motions for virtual characters. Our system is capable of real-time animations for complex body kinematics without encoding the expensive physics equations of motion. We conducted experiments on the use of the two proposed methods to gain insights into the most efficient and optimal control method for adaptive motion animations.

Despite the GA controller being more efficient than the QL controller, the inherent reinforcement learning scheme of QL may be suitable for some applications that requires direct external feedback. From the test results, both controllers can search for optimal motion solutions for real-time applications. Within a constant amount of time, the GA control architecture shows more accurate motions than the results of QL.

Future work includes an investigation in incorporating sensory and muscle nodes into the skeleton design to achieve biomechanical feasible motions for virtual characters. Example applications of such characters can be found in sports and physiotherapy for motion visualization. Moreover, it will be beneficial to create adaptive characters that can learn from hidden data by using visual information. Also further research can occur on the performance of the models as well. Amongst many interesting topics, seeding the population with possible good guesses and changing population size from generation to generation can also be further investigated.

8. References

[ABB05] ALANKUS G., BAYAZIT A.A., BAYAZIT O.B.: Automated motion synthesis for dancing characters, In *Computer Animation and Virtual Worlds 2005, Volume 16, Issue 3-4*, Wiley, 259 – 271.

[ALP04] ABE Y, LIU C. K., POPOVIC Z.: Momentum-based parameterization of dynamic character motion. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (August 2004), 173-182.

[BH00] BRAND M., HERTZMANN A.: Style machines. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (*July 2000*)*, p.183-192.

[BPW93] BADLER N. I., PHILLIPS C. B., WEBBER B. L.: *Simulating Humans: Computer Graphics Animation and Control*, Oxford University Press, 1993.

[DYP03] DONTCHEVA M., YNGVE G., POPOVIC Z.: Layered acting for character animation, *ACM Transactions on Graphics (TOG)*, v.22 n.3 (July 2003), 409-416.

[E05] ENDORPHIN: NaturalMotion. Siggraph 2005, http://www.naturalmotion.com/pages/technology.htm.

[FV93] FIUME E., VAN DE PANNE M.. Sensor-actuator networks. In *Computer Graphics SIGGRAPH '93 Proceedings, volume 27*, 1993, 335-342.

[GT95] GRZESZCZUK R., TERZOPOULOS D.: Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction, *ACM Computer Graphics, Proceedings of SIGGRAPH'95* (August 1995), 63-70.

[HH98] HAUPT R.L., HAUPT S.E.: *Practical Genetic Algorithms*, John Wiley, Canada, 1998.

[HOL75] HOLLAND J. H.: *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press, 1975.

[IBDB01] ISLA D., BURKE R., DOWNIE M., BLUMBERG M.: A Layered Brain Architecture for Synthetic Creatures. In *Proceedings of the International Joint Conference on Artificial Intelligence,* Morgan Kaufmann, 2001.

[NKL*04] NAM S. W., KIM D.H., LEE I.H., CHOI I., CHIEN S.I.: Reactive and Characterized Motion Generation of Virtual Character, In *Intelligent systems and Control*, 2004, ACTA press.

[NM93] NGO J.T., MARKS J.: Spacetime Constraints Revisited, In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques,* Siggraph 1993, 335-342.

[SB98] SUTTON R. S., BARTO A.G.: *Reinforcement Learning: An Introduction*, MIT Press, Cambridge 1998.

[SF04] SZAROWICZ A., FRANCIK J.: Human Motion For Virtual People, In *International Conference on Computer Games: Artificial Intelligence, Design and Education, CGAIDE* 2004.

[SIM94] SIMS K.: Evolving Virtual Creatures, In *Computer Graphics Siggraph '94 Proceedings* (July 1994), 15-22.

[TGW04] TANG W., GATZOULIS C., WAN T.R.: Proactive Anticipatory Virtual Characters- An Ethology Approach. In *Proceedings of Applied Simulation and Modelling* (June 2004), 172-177.

[WT02] WAN T. R., TANG W.: Learning by experience - autonomous virtual character behavioural animation, In *Proceedings of Intelligent Agents for Mobile and Virtual Media* (2002), 89 – 100.

[ZMCF05] ZORDAN V. B., MALKOWSKA A., CHIU B., FAST M.: Dynamic response for motion capture animation. *ACM Transactions on Graphics (TOG)*, v.24 n.3 (July 2005), 697-701.

[ZV03] ZORDAN V. B., VAN DER HORST N. C.: Mapping optical motion capture data to skeletal motion using a physical model. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (July 2003), 245-250.