공학박사학위논문

Interactive Control of Unorganized Human Motion using Deep Neural Networks

심 신경망을 이용한 조직화되지 않은 인간 동작의 실시간 컨트롤

2019년 8월

서울대학교 대학원

전기.컴퓨터 공학부

이 경 호

Abstract

Manipulating a virtual character is one of the major problems in computer graphics. The primary purpose of the character control is how to make virtual characters behave like humans, and how to control them as we want. Motion capture techniques enable realistic reproduction of human motion. As the number of actions increases, a more amount of motion data is required, and it is necessary to segment and label the clips corresponding to each action. These classification tasks need to be minimized because they should be performed manually. It is also a difficult problem to deal with multiple actions in one controller, since the parameters for manipulating each operation may be different. To control a character in real-time, controllers must have fast computation time and low memory requirements at runtime.

In this thesis, we proposed methods to learn the controller from unorganized motion data using a deep neural network. Using the deep neural network enables low memory usage because it does not need to have training data at runtime. It also makes the real-time operation possible using the GPU acceleration technique. We used Recurrent Neural Networks to capture underlying structures of human motion and also provide precise control over interactive characters. We proposed two techniques for manipulating virtual characters. The first is to manipulate the various actions on a single controller based on spatiotemporal constraints. We infer a character 's intention from learning motion data, and the user only needs to label the action frame of each motion. We also present a new data augmentation method that allows the model to be learned even from a small to moderate amount of training data. The second is to manipulate characters based on the user's performance. Performance-driven control allows users to perform intuitive control without having to learn how to operate them. However, the virtual character may follow unskilled movements of the novice

user, resulting in awkward motions. We propose a controller model that transforms the control parameters of the expert training data into values similar to those of the novice user. Our model can output a professional motion even from a beginner user. To verify the effectiveness of the proposed model, we showed the results using various motions such as basketball, tennis, dancing and walking.

Keywords: Computer Graphics, Data-driven Animation, Character Control, Interactive Control, Performance-driven Control, Machine Learning, Deep Learning, Recurrent Neural Network

Student Number: 2012-20823

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Character animation is a technique for expressing the continuous movement of a virtual character. In the early days, a professional animator created the movement of the character for each frame by drawing it directly. In recent years, motion capture techniques have been introduced that use real human motion to reproduce more realistic human movement. Although motion capture is intended to replicate a person's actual motion, if a large amount of motion data is prepared in advance, it may be used to create a new motion. This is the basic concept of data-driven character animation, which uses mocap database to create realistic and natural motion.

Character control problem is to create a motion of a virtual character that moves according to the user's input. The input of the character control includes spatio-temporal conditions such as "where to go", "when to arrive", "what to do", and "how to move". The virtual player may be controlled, for example, to go to the corner of the basketball court, pass the ball to another player, receive the ball back, and shoot the ball to the rim. Main challenges of the character control are how to make virtual characters behave like humans, and how to control them as we want. Humans move

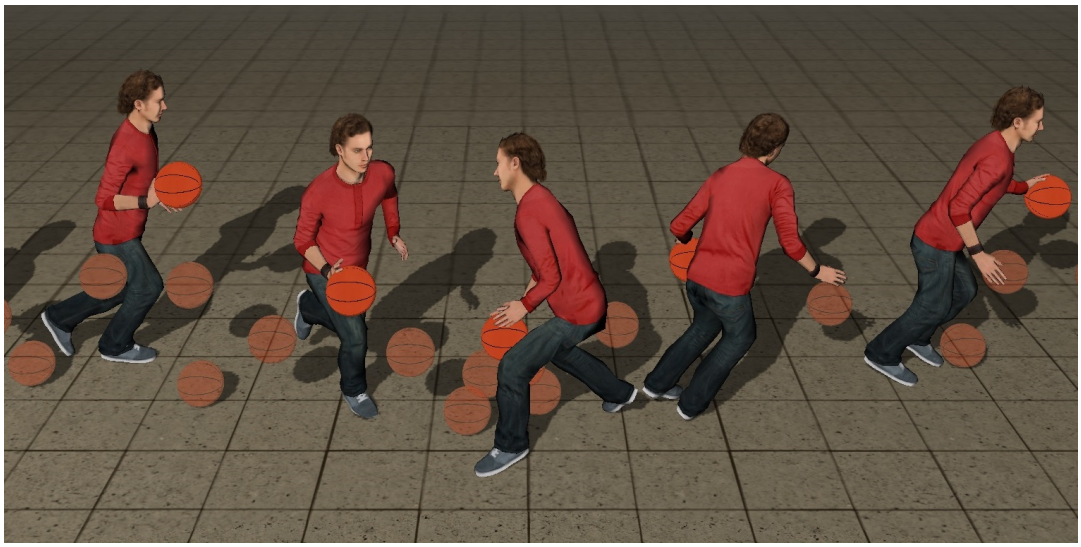

Figure 1.1 An example of controlling a virtual character to play basketball.

in diverse, but highly structured ways. Various factors, such as habitual patterns, social customs, sports rules, and dance choreography, contribute to the formation of behavioral structures. Capturing, modeling, and reproducing such a structure for virtual characters is a challenging problem.

There are some other technical issues in learning the character controller. In the data-driven approach, the size and quality of the training data have a significant impact on the quality of the results. The controller should be able to handle large amounts of data. In order to use captured motion, it is necessary to segment and label each action. These manual works take a lot of time and effort, and the controller should minimize these preprocessing. Interactivity is another crucial factor. To control a character in real time, controllers must have fast computation time and low memory requirements at runtime.

Despite many studies in this field, no approach has been proposed to address these issues yet completely. Finite state machine is a method widely used in the computer

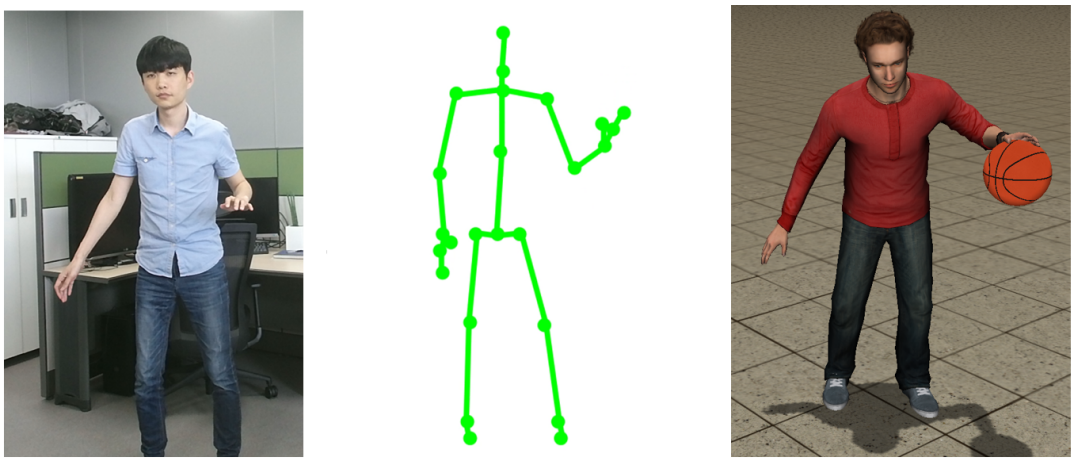

Figure 1.2 Example of Performance-driven character control.

game field, which is a method of segmenting each action and specifying the connection between each movement. This allows fast, high-quality control at the runtime, but it requires a lot of manual work to model the state machine. The search-based method of finding the transition between motions according to a given input can handle complex tasks, but it is challenging to apply it to online because the computational cost is too large. Recently, a deep neural network has been studied as a controller model that can be easily learned while having memory efficient and fast performance at runtime. However, it still only deals with cyclic or simple movements such as locomotion.

The main goal of this thesis is to learn various motions on a single controller with minimal manual efforts. Although large motion databases are readily available, the motion data only record the actions taken by an actor but his/her intention is not explicitly represented. Our learning model infers the intention of action from raw motion data to build rich connections between a multitude of control objectives and a large repertoire of actions. Multiple actions can have a different set of control parameters. We present a new multi-objective control model that can handle various

control parameters on a single controller. Performance-based control is another way to control a virtual character(see Figure 1.2). Using the user's actions directly as control parameters is an efficient and intuitive method. However, if the controller follows the behavior of the novice user, it can generate awkward motion. We propose a new control parameter adaptation technique that allows the controller to output natural movement from the user's unskilled action.

Interactive character control with multiple objectives: We present an approach that learns to act from raw motion data for interactive character animation. Our motion generator takes a continuous stream of control inputs and generates the character's motion in an online manner. The key insight is modeling rich connections between a multitude of control objectives and a large repertoire of actions. The model is trained using Recurrent Neural Network conditioned to deal with spatiotemporal constraints and structural variabilities in human motion. We also present a new data augmentation method that allows the model to be learned even from a small to moderate amount of training data. The learning process is fully automatic if it learns the motion of a single character, and requires minimal user intervention if it deals with props and interaction between multiple characters.

Performance-driven character control with parameter adaptation: Recently, the development of a low-cost motion sensor has enabled the user to control the motion of a virtual character directly. We proposed a new learning method that can generate the natural movement of the expert from the user's unskilled motions. Our model consists of two parts, input parameter perturbation and parameter distribution prediction model. The first part adds deliberate noise to the control parameter at the training time, to generate a mapping between awkward control parameters and ex-

pert motion. The second part learns the distribution of the acceptable range of control parameters from the current motion sequence. This serves to project the immoderate parameter into the appropriate range. These two models provide a stable connection between the control parameter and the output motion, allowing natural movement can be output even for the strange inputs. Our control parameter adaptation model is fully automatic, it can learn the controller without any manual preprocess.

Chapter 2

Background

2.1 Data-driven character animation

In this section, we will review data-driven approaches studied in computer graphics. The key idea of data-driven approaches is to utilize a collection of high-quality motion data to generate new motions potentially with new target characters and new target environments. The early work focused on building data structures or computational models of human movements to support efficient motion generation, search, and manipulation. The popular idea based on *motion graphs* allows transitioning between motion frames to generate a novel sequence of motions [1, 2]. Temporally aligning and blending a collection of similar motions can generate a family of parametrized motions [3, 4]. Alternatively, a collection of similar motions can be used to build a statistical model based on principle component analysis [5] and Gaussian Process Latent Variable Models (GPLVMs) [6, 7]. The motion graph is equivalent to a finite state machine, if we consider each individual motion frame (or motion segment) as a symbol. The expressive power of a finite state machine is limited and thus can de-

scribe only very simple structures in diverse human behaviors. Hyun et al [8] studied *motion grammars*, which are essentially context-free grammars augmented with spatiotemporal conditions, to model complex behavioral structures. Motion grammars and the syntax of human activities have also been studied in the context of computer vision, linguistics, and robotics [9, 10].

2.2 Character control techniques

Directing a character to move to a goal position or along a target trajectory requires appropriate control and/or planning algorithms together with the computational model of human movements. Character animation has often been formulated as state space search in high-dimensional configuration space and addressed using the $A^\star$ algorithm or its variants [1, 2, 11]. State space search based on branch-and-bound is notorious for its heavy computational burden, which can be mitigated by various types of precomputation on state space, action space, and environment configuration space. Lee and Lee [12] introduced reinforcement learning methods into computer graphics community. The key idea of reinforcement learning is to learn state value functions during the learning phase such that the optimal action at each individual state can be determined immediately at runtime without state space search. Treuille *et al.* [13] employed linear function approximators to compactly represent state value functions. The revolutionary development of deep learning replaced linear function approximators with deep neural networks to achieve dramatically improved representation power and learning capacity [14]. The problem can alternatively be formulated from the viewpoint of path planning. Popular path planning algorithms using probabilistic roadmaps and rapidly-growing random trees have been exploited for full-body character animation in virtual environments with obstacles and dynamic environment

features [15, 16].

Spatiotemporal conditions play a key role of simulating multi-character interactions. Various approaches have been studied to spatially align and temporally synchronize the motion of multiple characters interacting with each other. *Motion patches* [17] were originally invented to simulate character-object interactions. A motion patch is a fragment of character animation that captures how a character interact with an environment object. Given a collection of motion patches, character animation is simply a spatiotemporal tiling of motion patches. The concept of motion patches have evolved to deal with interpersonal interactions as well [18, 19]. Multi-character interactions can be specified through various forms of user interfaces. Won *et al.* [20] translated a drama script into an *event graph* and used a generate-and-rank approach to synthesize animated scenes in which multiple characters interact with each other. Hyun *et al.* [8] generated the animation of basketball plays from simple drawings on a tactic board. The previous studies in this category using a collection of canned motion segments and/or motion patches commonly formulated the problem as mixed optimization of combinatorial planning and continuous optimization. Deciding the sequence of motion segments is a discrete, combinatorial problem, while precisely aligning spatial locations and timings is a continuous optimization problem. The mixed optimization has previously been addressed using stochastic methods, such as simulated annealing and MCMC sampling, which are computationally expensive. The previous studies reported that animating dozens of characters for 10 to 20 seconds takes several minutes to several hours of computation on a typical PC.

2.3 Performance-based character control

A performance-based control takes the user's motion as a direct control input. In this case, the higher expressive power of the input device, the more efficient and intuitive control is possible. Shiratori and Hodgins [21] used a three-axis accelerometers as input devices, and Ishigaki *et al.* [22] introduced optical motion capture system as a control interface. Recently, single depth camera such as Microsoft Kinect [23] is popularly used as motion sensor. Mehta *et al.* [22] suggested a pose estimation system with single RGB camera.

In performance-based control, the user can control to a detailed pose of the character, which is efficient for handling complex motions such as a dance motion. Raptis *et al.* [24] proposed a real-time gesture classification system for dance motion. It has pre-defined dance motion set, and classifies input motion to one of the action set. Tang *et al.* [25] implemented an interactive dancing game using optical 3D motion capture system.

Following the novice user's actions can cause the controller to output an unnatural motion. Ishigaki *et al.* [22] adopted physics simulations to generate natural motions. Their system leverages a physical simulation, a set of offline action examples, and an optimal integration process to synthesize the character's motion. Lee and Kwon [26] parameterized over a set of modulated reference motions that aims to cover the range of possible user actions. Mousas *et al.* [27] proposed a hybrid control that lies within the performance animation and the motion controller. These approaches are focused on limited action sets or simple motions such as locomotion. Our goal is handling more complex motions such as dance or fight action, without any pre-defined action set or heavy labeling processing.

2.4 Machine learning algorithms

A new class of machine learning algorithms exploiting the expressive power of deep neural networks have successfully been applied to character animation. Holden *et al.* [28] learned biped locomotion along target root trajectories using Convolutional Neural Networks (CNNs). In the next year, they utilized a fully-connected neural network with a latent phase variable to interactively generate biped locomotion on uneven terrain. The latent phase variable is useful to align cyclic movements in the training data and thus accelerate the progress of learning. Zhang *et al.* [29] generalized the phase-functioned architecture further to deal with multiple modes, which allow the model to learn various gaits in quadrupled locomotion. Deep reinforcement learning that incorporates deep neural networks into the framework of reinforcement learning as function approximators has been particularly successful for physically based simulation and control of biped locomotion [30–33] and flapping flight [34]. Tan *et al.* [35] used feedforward neural networks to physically simulate bicycle stunts.

Recurrent Neural Networks (RNNs) are popular models for processing sequential, time-series data. The RNN takes inputs, update its internal state through recurrent connection, and generates outputs at every time-step iteratively. Therefore, the history of inputs affects the generation of outputs. The online and history-dependent nature of the RNN makes it particularly suitable for natural language understanding, speech recognition and machine translation [36, 37]. It was reported that a variant of RNNs, Long Short-Term Memory (LSTM) networks, are capable of learning simple context-free and context-sensitive languages [38]. Recently, the RNNs have also been used for image processing [39, 40] and character animation [41]. The RNNs learned from full-body human motion data demonstrated their effectiveness successfully in action recognition [42] and action prediction [43–47]. We are interested in motion

synthesis for interactive graphics applications, which poses new challenges for RNN-based approaches.

Chapter 3

Interactive Character Control with Multiple Objectives

3.1 Overview

Interactive character animation is an important issue in computer graphics. Among the many technical aspects of character animation, we are particularly interested in two key challenges: how to make virtual characters behave like humans, and how to control them as we want. Humans move in diverse, but highly structured ways. Various factors such as habitual patterns, social customs, sports rules and dance choreography contribute to the formation of behavioral structures. Capturing, modeling and reproducing such a structure for virtual characters is a challenging problem. For example, the movement of our virtual basketball player is governed by basketball regulations, which describe how many steps the player can take while holding a ball, when pivoting is allowed, and so on.

Interaction with virtual characters includes specifying spatiotemporal conditions such as "where to go", "when to arrive", "what to do", and "how to move". The virtual

player may be controlled, for example, to go to the corner of the basketball court, pass the ball to another player, receive the ball back, and shoot the ball to the rim. Given the sequence of conditions, the virtual player has to plan its actions in compliance with basketball regulations. The problem becomes even more challenging if multiple players have to collaborate and synchronize their movements while interacting with each other. This problem has previously been addressed using either state space search [1, 2, 11] or mixed discrete-continuous optimization [8, 18, 20]. Both approaches are computationally demanding and thus cannot achieve interactive performance.

Recently, deep learning and neural networks have received a great deal of attention as a means of modeling, predicting, recognizing, and synthesizing human movements [29, 43, 46, 48, 49]. Learning to act by interacting with the environment in the framework of interactive character animation has not been fully addressed yet. Issues of technical interest include dealing with a diversity of action repertoires, a multitude of dynamically specified goals, and continuous control over space and time.

In this paper, we present a new supervised learning approach that takes a continuous stream of control inputs to generate character animation interactively. The common practice of supervised learning requires a collection of example input-output pairs. In our case, the input corresponds to the intention (or objective) of action while the output is the actual motion the character takes. Although large motion databases are readily available, the motion data only record the actions taken by an actor but his/her intention is not explicitly represented. Our learning model infers the intention of action from raw motion data to build rich connections between a multitude of control objectives and a large repertoire of actions.

It is well-known that Recurrent Neural Network (RNN) is remarkably adept at modeling sequential data. Our motion generator is a multi-layered RNN conditioned

to deal with spatiotemporal constraints and rich structural variabilities in human motion. Our experiments demonstrate that spatiotemporally conditioned RNN can not only capture underlying structures of human motion but also provide precise control over interactive characters. Our approach shares a number of advantages with existing data-driven, supervised learning approaches [29, 49]. Our motion generator is suitable for real-time interactive applications since it generates motion frames one-by-one in an online manner without delay. The motion generator is compact since the training data can be discarded after learning. Secondary motion effects and props can also be learned together with the character's full-body motion.

We also present a new data augmentation method that allows our model to be learned even from a small amount of training data. The augmentation method systematically enriches spatial, temporal, and combinatorial variabilities in the training data. The augmentation process is fully automatic with the motion of a single character, while the process requires minimal user intervention to deal with props and interaction between multiple characters.

3.2 Multi-Objective Control Model

Consider a character that interacts with the environment over discrete time steps. At each time step t, the character receives an observation and makes a move to update its full-body pose and location for the next time step. The observation in three-dimensional simulations is rich including the full-body pose of the character, the configuration of the environment, and the configuration of its collaborative/adversarial counterparts. Any desired future observation can serve as a control parameter. The user may specify where and when the character will execute a target action, and can also specify the quality and style of the generated motion.

The character may have a set of actions it can choose from. For example, the basketball player can perform actions such as *Dribble*, *Shoot*, *Pass*, and *Catch*. Each action is parameterized by a different set of control parameters $\mathbf{f}_a$, which is a subset of the full observation. For multi-objective control, we use integrated control parameters F as input of the system.

$$F = \bigcup_a f_a,$$

At each time step t, the user provides control parameters $\mathbf{f}_a$ specific to the action currently performing. We want to leave the parameters irrelevant to the current action unspecified. However, it is difficult to implement unspecified values in the network-based architecture. Instead, we predict those irrelevant parameters from the current states using the control network to feed them back into the network. The prediction is the future observation the system will likely to achieve assuming that there is no intention (or control) to regulate the future states.

The control task is to decide a next move at every time step t that best achieves the goal

$$G_t = \text{diag}(S)\ F_t + \text{diag}(1 - S)\ P_t,$$

where S is a binary vector with its entries either 0 or 1. The selection vector is defined for each individual action type to select entries relevant to the particular action from F_t. Our multi-objective control model takes a dynamically specified target G_t and current state $\mathbf{m}_t$ as input and generates a next state $\mathbf{m}_{t+1}$ and the prediction of control parameters P_{t+1} as output. Interactive control is a discrete-time process of generating motion $M = \{\mathbf{m}_t\}$ starting from initial configuration $\mathbf{m}_0$.

Full-body Configuration. The animated character in motion capture data is an articulate figure of rigid segments connected by ball-and-socket joints. The motion of an articulated character is represented by its time-varying position and orientation

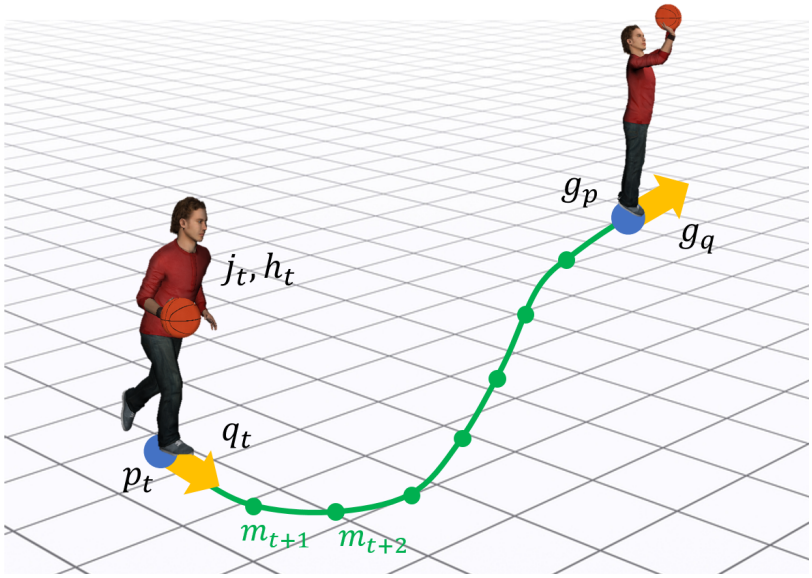

Figure 3.1 Interactive character animation using user control inputs.

of the root body segment in the reference coordinate system together with its joint angles also varying over time. It is often beneficial to maintain both joint angles and joint positions together in training networks even though the representation is redundant. Let $\mathbf{m} = (p, q, \mathbf{j}, \mathbf{h}, \mathbf{c})$ be the full-body configuration of a character, where $p \in R^2$ is the position of the character, $q \in R^2$ is its facing direction, $\mathbf{j}$ is a long vector concatenating joint angles, and $\mathbf{h}$ is another long vector concatenating joint positions. Physical contact (e.g., foot-ground and hand-ball contact) is yet another important information to understand the physical state of human poses. $\mathbf{c}$ is a binary vector that encodes whether individual body parts, such as forefeet and heels, are in contact with the ground surface or any props.

User Control. One way of controlling a character is to specify target position $g_p \in R^2$ and facing direction $g_q \in R^2$ interactively and command the character to track the target (see Figure 3.1). Additionally, we can specify which action g_a to perform

at the target location and how long it takes to get there. Let $U = (g_p, g_q, g_a, g_t)$ be a user control input, where g_a is a one-hot vector denoting an action label and g_t is the timing of action. various control parameters, such as moving direction and velocity, can be added to the target list if necessary. The flexibility of user control is important, because our model represents various types of actions and each type may require a different form of user control. Here, all spatial coordinates are represented with respect to the body local coordinate system of the character at time t and all temporal values are relative to the current time t.

Quality Attributes. The quality attribute describes a certain aspect of movements that can be measured from motion data. For example, *Straightness* describes how straight the moving path is, and *Uprightness* describes the posture of the character in action (see Figure 3.2). The individual attribute at time t is a scalar value of $[0, 1]$ and evaluated from a window of neighborhood $(\mathbf{m}_{t-\tau}, \cdots, \mathbf{m}_{t+\tau})$. The quality attribute also serves as a control parameter to specify the style of the generated motion.

Bounded parameters. The control vector includes many parameters, and each parameter will be normalized to have a mean of zero and a standard deviation of one for network training. Therefore, in this network-based formulation, there is no notion of weighing the significance of one parameter with respect to the others. Instead, we use a different approach to modulate the significance of individual parameters. For example, the range of an orientation parameter is bounded within a 2π-interval (or [0,1] if unit quaternions are used), while the range of a position parameter is unbounded. Consider the normalized value of a target position g_p, which is discriminative when the target is far away (large value). As the character approaches the target, the magnitude of the parameter becomes small and not discriminative anymore. To cope with

Figure 3.2 Example of Quality Attributes. The control task requires the agent to move from one position to the other in time T with *Straightness* attribute S, and T is longer than it normally takes for the agent to complete the task. (Top) If S is close to one, the agent will walk slowly along the straight line between the two positions. (Bottom) Otherwise, if S is close to zero, the agent will take a curved path at normal speed.

this range resolution problem, we may use an auxiliary bounded parameter

$$\bar{g}_p = \min(g_{\max}, \|g_p\|)\frac{g_p}{\|g_p\|},$$

where $g_{\max}$ is the bound of $\bar{g}_p$. $\bar{g}_p$ is discriminative only when the target is close within the bound. g_p and its bounded value $\bar{g}_p$ operate together as a coordinated pair of long-range and short-range sensors. Auxiliary bounded parameters can be exploited for any type of parameters to supplement their resolution power.

3.3 Network Training

RNN is a kind of neural networks particularly useful for modeling sequential data. Unlike many feedforward neural networks, a RNN maintains hidden internal states that is not only dependent on the current observation, but also relies on the previous hidden state and hence the previous observations. RNN Training is similar to feedforward network training in the sense that network parameters are updated incrementally via backpropagation. Since parameters are shared by all time steps in RNN, gradients at the current time step would affect gradient computation at the previous time steps. This process is called Backpropagation Through Time (BPTT). We built our RNNs using Long Short-Term Memory units (LSTMs), which preserves gradients well while backpropagating through time/layers and thus can deal with long-term dependencies.

Our network model consists of multiple LSTM layers with encoder and decoder units (see Figure 3.3). The input to the network is the current full-body configuration $\mathbf{m}_t$ and a goal G_t. The output is the prediction of control parameters $\hat{P}_{t+1}$ and the full-body configuration $\mathbf{m}_{t+1}$, which recursively feeds back into the network at the next time step.

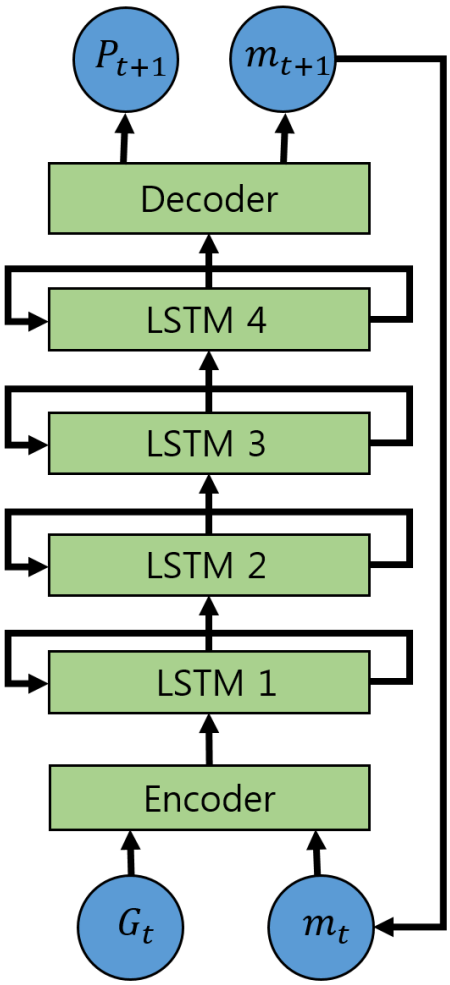

Figure 3.3 Network architecture.

3.3.1 Preprocessing Training Data

Our network is trained using raw motion capture data, which are labeled with minimal manual effort. The training data are often unstructured consisting of a number of short motion clips. Manual labeling is needed when there is a prop or an interaction between multiple characters. We selected and labeled key frames in which a key action occurs. Any frame without explicit action label has default action, which is Move-To in our experiments. The key action usually involves the change of contact states, such as a ball leaving the hand, a racket hitting a ball, and a dancer putting his/her hand on the shoulder of the other. Foot-ground contact can be identified automatically if the foot is close to the ground surface and its velocity is below a certain threshold. Ball-Hand and Ball-Racket contacts are not easy to identify since the ball is not included in motion capture data. We labeled Ball-Hand and Ball-Racket contacts semi-automatically throughout the training data.

3.3.2 Control Objectives

The common practice of supervised learning requires a collection of input-output pairs and searches for network parameters that minimize the desired and simulated outputs given input data. Unlike this common practice, our training data do not have such input-output pairs, but only have a collection of motion sequences that correspond to the outputs of the neural network. Fortunately, task descriptions can be inferred from training data to build input-output correspondences. This inference step allows RNN to be learned even if we do not know how to create desired outputs given arbitrary inputs.

RNN training is a randomized process. A motion clip is selected randomly from a pool of training data. If the motion clip is very long, a random subsequence of fixed

length (in our experiments, the subsequences are 6 seconds long) is selected from the clip. Let $\hat{M} = \{\hat{\mathbf{m}}_t\}$ be a sequence of motions. We use a hat above a symbol to indicate the measurements in the training data. The task $\hat{G}_t$ at every time step t should be inferred from the data. We assume that the agent has a multitude of intentions (or objectives) at every moment. Specifically, given a set of temporal offsets $\tau_1, \cdots, \tau_n$, the agent at time step t has multiple objectives of performing the action labeled at frame $\hat{\mathbf{m}}_{t+\tau_i}$ in time τ_i for any $1 < i < n$. This approach generates a dense set of control tasks to be learned by the network. The rich multi-objective learning allows precision and flexibility of control with a variety of actions in the unstructured training data.

The total amount of information RNN has to learn depends on the time horizon it wants to predict. RNN should be bigger and deeper to look ahead further, since many possibilities that can happen in the lookahead interval have to be learned and encoded. In our implementation, the time horizon $t_{\max}$ is limited to 4 seconds.

3.3.3 Loss Function

All consecutive pairs $(\hat{\mathbf{m}}_t, \hat{\mathbf{m}}_{t+1})$ are used to evaluate the loss function, which has three terms:

$$E = E_{\text{motion}} + E_{\text{contact}} + E_{\text{rigid}}. \tag{3.1}$$

E_{motion} penalizes the discrepancy between training data and network outputs.

$$E_{\text{motion}} = \sum_t \|\mathbf{m}_t - \hat{\mathbf{m}}_t\|^2, \tag{3.2}$$

where $\hat{G}_{i-1}$ is the input to the network, $\hat{\mathbf{m}}_t$ is the expected output observed in the training data, and $\mathbf{m}_t$ is the actual output from the network. E_{contact} penalizes foot sliding.

$$E_{\text{contact}} = \sum_t \sum_{k \in \text{Feet}} c_t^k c_{t+1}^k \|\mathbf{h}_t^k - T(\mathbf{h}_{t+1}^k)\|^2, \tag{3.3}$$

where k is the index of a joint position. Here, c_t^k is 1 if joint k is in contact with the ground surface at time step t. Otherwise, $c_t^k = 0$. $T(\mathbf{h}_{t+1}^k)$ transforms the coordinates of $\mathbf{h}_{t+1}^k$ to compare with $\mathbf{h}_t^k$ while taking body translation and rotation at $[t, t+1]$ into consideration. Condition $c_t^k = c_{t+1}^k = 1$ indicates that the contact remains for the time step. E_{rigid} penalizes the lengthening and shortening of body links.

$$E_{\text{rigid}} = \sum_t \sum_{(k,m)\in\text{Links}} (\|\mathbf{h}_t^k - \mathbf{h}_t^m\| - l_{km})^2, \tag{3.4}$$

where joint k and joint m are adjacent to each other connected by a rigid link and l_{km} is the length of the link. The distance between the adjacent pair of joints should be preserved.

3.4 Learning Ball Trajectory

Interactive character animation often includes props, such as balls and rackets. Characters and props move together in a synchronized manner. Dribbling the basket ball is of particular importance, because the ball goes through interesting physical stages: ball-in-hand, ballistic-flight, and bounce-on-ground. The ball in hand moves synchronized with the body. When the ball is thrown, the ballistic flight is governed by a physics law and its moving trajectory is independent of the character's motion. Bouncing on the ground yields a rapid change of the moving direction. We consider the ball as an extra joint of the articulated figure and learn dribble actions including the ball.

Our basketball training data do not include balls, so we synthesized ball trajectories via physics simulation. Since ball-hand contacts are annotated at frames, the initial position and velocity of the ballistic flight, and its bouncing point can be estimated from motion data. We augmented training data with simulated ball trajectories. The ball position b_t is described with respect to the body local coordinates

and included as part of the character's pose.

Even though RNN is adept at learning ballistic trajectories, extra loss terms are required to prevent visual artifacts. E_{ball} is added to the loss function of the full-body motion in Equation (3.1).

$$E_{\text{ball}} = E_{\text{hand}} + E_{\text{bounce}}, \tag{3.5}$$

where E_{hand} keeps the ball in hand when the contact flag $c_t^k = 1$.

$$E_{\text{hand}} = \sum_t \sum_{k \in \text{Hands}} c_t^k \| |\mathbf{h}_t^k - b_t| - \text{radius} \|^2. \tag{3.6}$$

The distance between joint $\mathbf{h}_t^k$ and ball b_t is maintained to be equal to the radius of the ball. E_{bounce} enforces that the ball bounces at the ground height. Let c_t be the contact flag that goes on if the ball touches either hands. The contact flag indicates that the ball is pushed off when $c_t = 1$ and $c_{t+1} = 0$. The ball travels and bounces on the ground while $c_t = c_{t+1} = 0$. The ball comes back to a receiving hand when $c_t = 0$ and $c_{t+1} = 1$. Assuming that the trajectory is ballistic, the minimal height of the ball between push-off and receive-back is its height when it bounces on the ground. E_{bounce} prevents the ball from bouncing before it reaches the ground height.

$$E_{\text{bounce}} = \sum_t (1 - c_t) c_{t+1} \| y_t - \text{radius} \|^2, \tag{3.7}$$

where y_t is the minimal height of the ball since the last push-off. y_t is updated incrementally at every frame.

$$y_{t+1} = c_t b_{t+1}^y + (1 - c_{t+1}) \min(y_t, b_{t+1}^y), \tag{3.8}$$

where b_t^y is the y-coordinate (height from the ground surface) of the ball at time t. Note that bouncing happens in the middle of ballistic flight and the loss is reflected later when the ball comes back to the hand. Backpropagation through time allows such a delayed loss to be reflected in network training.

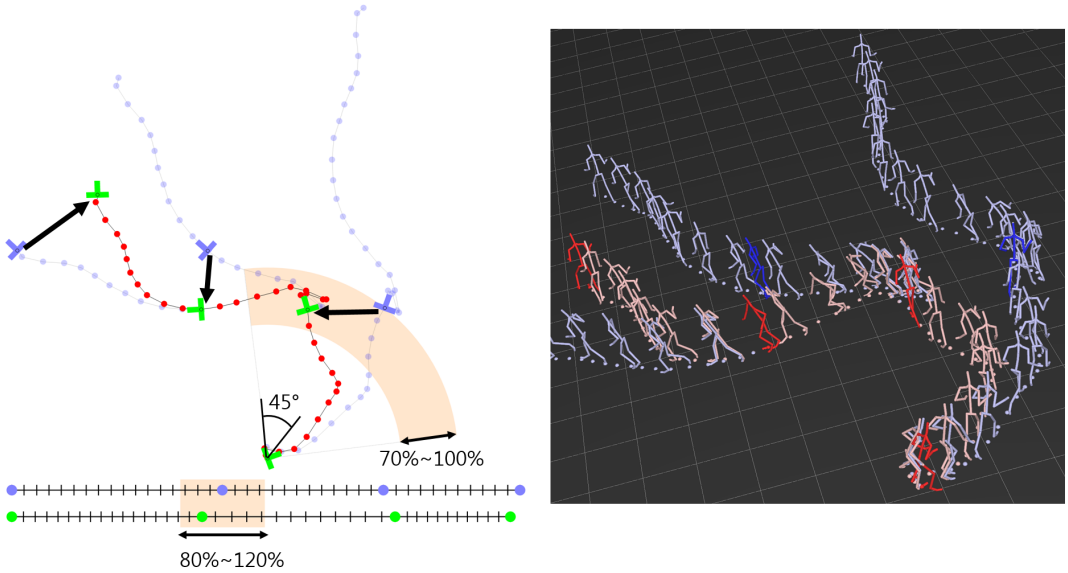

Figure 3.4 Spatial and temporal perturbation of motion data. The spatial trajectory depicted in blue is perturbed to generate spatial variations. A varied spatial trajectory is depicted in pink. Temporal variations are generated by varying the timing of actions (left, bottom).

3.5 Data Augmentation

The performance and accuracy of our model depends mainly on the size and quality of training data. The amount of high-quality motion data available in each individual application domain is insufficient. In our examples, the size (playing time) of raw motion data for each domain ranges from small (only a few seconds) to moderate (20 minutes). We augment the raw data to increase both combinatorial and spatiotemporal variations. The whole process is fully automatic if the underlying behavior is simple in the class of *regular language*, and semi-automatic if syntactic structures in the class of *context-free language* are required.

3.5.1 Simple regular behavior

We begin with simple behaviors that can be captured by a finite state machine. The learning process is episodic. The goal of data augmentation is to create a large number of episodic motion sequences, exhibiting a diversity of action sequences, moving directions, and speeds. Given a collection of raw motion data, we generate episodes through two steps. The first step is to generate random combinatorial variations by resequencing motion frames. To do so, we constructed a motion graph from the input data [1]. The motion graph is a directed graph that takes motion frames as its nodes. The edge of the graph represents a transition probability from one frame to the other. The transition happens more likely if the start and end frames are similar to each other. Random walk through the graph generates a novel sequence of motions by string visited frames together. The random walk is discrete and combinatorial since only discrete choices (transitions) are provided at every node. The length of random walk depends on the complexity of behavioral structures we want to capture from the data set.

The second step is to enrich spatial and temporal variations by perturbing motion sequences from random walk. Simply adding Gaussian noise could introduce visual artifacts and corrupt the quality of the output motion. Perturbation should be conducted while preserving the integrity and coherence of human motion. To do so, we employ Laplacian motion editing [50], which can deform the spatial trajectory and timing of motion data in an as-rigid-as-possible manner subject to user-provided positional, directional, and timing constraints (see Figure 3.4). Given a motion sequence from random walk, we first divide it into segments of random lengths, then deform the spatial trajectory of each segment randomly in the range of its length from 70% to 100% and its angle from -45°to 45°, and finally connect those segments back together

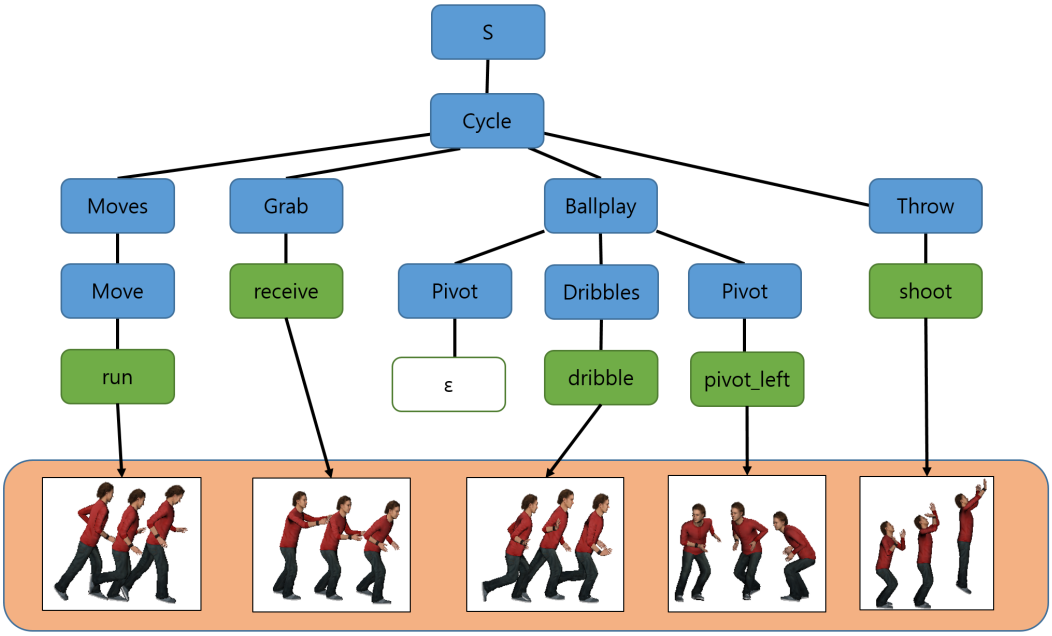

Figure 3.5 Example of grammar expansion

to have the spatial trajectory of the whole sequence perturbed smoothly. We also add timing constraints at random time instances to accelerate or decelerate randomly in the range of 80% to 120% of the original speed.

3.5.2 Context-free behavior

While motion graphs capture simple structures that can be inferred immediately from observations, more complex behavior patterns are required for various character animation applications. Ideally, we wish to learn a formal grammar from training data via grammar induction. However, motion databases currently available for individual domains are insufficient for automatic grammar induction. Instead, we provide with a manually-crafted formal grammar (see Appendix).

The use of a motion grammar requires extra efforts for motion segmentation and labeling. The basketball regulation requires detailed action classification. We defined

motion grammars similarly to the work of Hyun *et al.* [8]. The motion grammar has a finite number of terminal and non-terminal symbols, and includes a set of production rules. Each terminal symbol (a.k.a. action symbol) represents an action and may corresponds to multiple motion segments. For example, we have many Jump-And-Shoot motions in the training data. All of the motions match to the same action symbol. One of the nonterminal symbols serves as a starting symbol and fully expanding non-terminal symbols produces a string of action symbols. One non-terminal symbol may have multiple production rules. Randomly applying production rules generates a family of random parse trees representing diverse structural variabilities. Given a string of action symbols, selecting motion segments corresponding to individual action symbols and concatenating them will generate a new episodic motion sequence, which goes through the process of spatial and temporal perturbation in Section 3.5.1.

3.6 Experiments

We implemented our algorithm in Python using TensorFlow [51] for the learning and evaluation of recurrent neural networks. The computing power of GPUs (NVIDIA GeForce GTX 1080) were utilized to accelerate neural network operations. We use multi-layer RNNs with four LSTM layers of size 512. The time step for truncated BPTT is 48. Our implementation uses tanh activation function and Adam optimization algorithm. The network architecture and the network output format are the same in all examples, but the network input depends on the application domain and the design of control parameters. The largest network model takes 32 Mbytes in main memory. Learning takes about 12 to 24 hours on a single GPU. Each dimension of input and output vectors is normalized to have zero mean and one standard deviation for network training, and denormalized to compute loss E_{contact} and E_{rigid} in

Cartesian space.

We utilized a collection of motion data sampled at the rate of 30 fps. The total playtime of the basketball, tennis, and locomotion data are 10 to 12 minutes long. We built a motion graph for each data set, and data augmentation generated 4 hours of motion data for each domain. The motion grammar was constructed only for the basketball example (see Appendix for the motion grammar). The learning process is fully automatic except for the tennis and basketball examples. The tennis data require minimal manual processing to label the motion frames in which the racket hits the ball. The basketball data set require manual efforts for motion segmentation and labeling. The detailed information about the training data is summarized in Table 3.6.

Name	**Motion Length**	**Training Time**
Basketball	12 minutes	24 hours
Tennis	10 minutes	24 hours
Gorilla	10 seconds	12 hours
Drunken	30 seconds	12 hours
Zombie	1 minute	12 hours
Dance	1 minute	12 hours

Table 3.1 The size of the training data and training time.

Locomotion The locomotion example demonstrates an interactive character that can be controlled with only a few control parameters. The training data include unstructured motion clips exhibiting walking, turning, side stepping, and backward stepping. The user can specify either target position or target direction, or both to

control the character interactively. Being able to enable/disable control options is an important aspect of multi-objective control that allows many, unorganized action repertoires to be learned in a single network model. The network model learned rich and temporally dense supervision that enables multi-objective control. We use quality attributes to select particular gaits or locomotion styles. Each attribute can be either binary or continuous scale in $[0, 1]$ depending on the composition of training data. If training data include distinctive gaits and styles without any transitioning between them, binary attributes select a particular one among them. If there are a lot of transitioning between gaits and styles in training data, continuous-scale attributes generate inbetween styles as if motion blends do.

Basketball In the construction of the basketball model, we define six key actions (*Shoot*, *Pass*, *Receive*, *Catch*, *Dribble*, *Walk*) and four control parameters (position, direction, action type, timing). Each action is parameterized by a combination of the control parameters. For example, *Pass* takes two control parameters (position and direction), which describe the location where the action will occur and the direction it throws the ball, respectively. *Receive* takes a different combination of control parameters (position and timing), which describe the location where the player receive the ball and the timing of the ball arriving the location, respectively. The user can control the character interactively by dragging and triggering key actions. The target position, orientation, and timing are determined appropriately when an action is triggered. The ballistic trajectory of the ball after shooting is physically simulated. *Catch* requires the character to predict the ballistic trajectory and decide where and when it will catch the ball. In the multi-character setting, the user controls one character with the ball at a time while the other characters move autonomously based on stochastic play patterns.

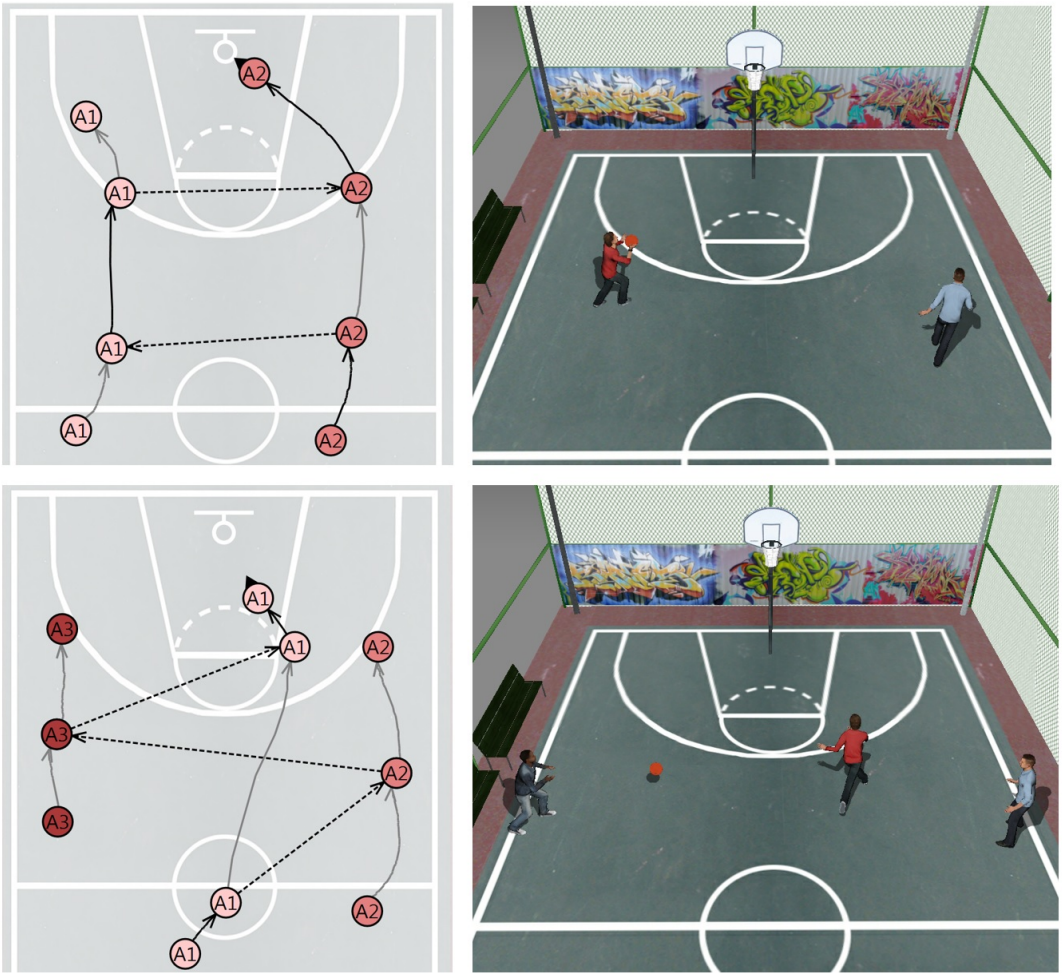

Figure 3.6 Animation authoring from drawings on the tactics board.

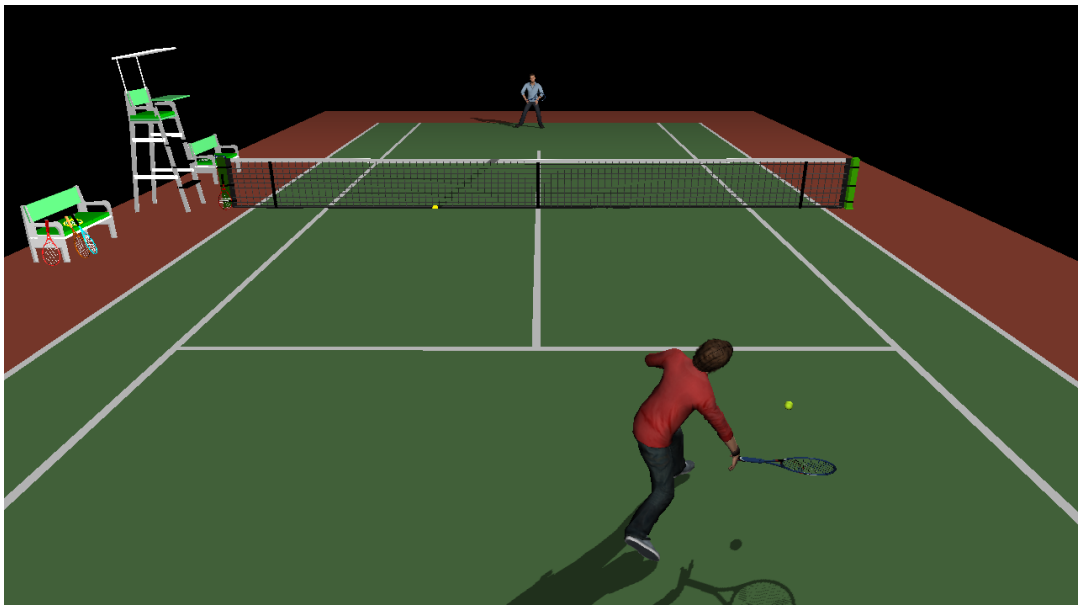

Figure 3.7 A snapshot of interactive tennis play.

Tactics Board Hyun *et al.* [8] demonstrated a tactics board interface, which allows a user to draw tactical plans on the board and generates the animation of 3D full-body basketball players acting out the plans. We reproduced their results using our network model (see Figure 3.6). The motion grammar and the MCMC algorithm by Hyun *et al.* took 4 and 6 minutes to generate animations, respectively. We took their motion data and motion grammar to learn our model. Single character motion generation with RNN takes only 5 millisecond per frame. The tactics board examples take 2.76 and 3.84 seconds, respectively. Our algorithm is about 45 times faster than the MCMC algorithm.

Tennis Two actions, *Stroke* and *Return*, are defined for the tennis example. The character strokes the ball and returns to the default location to prepare for the next stroke repeatedly. Control parameters can be defined flexibly depending on the design of a scenario and a user interface for character animation. Since the ball travels across

the court between two players, the ranges of the ball positions and timings (20 to 30 meters in space and 1 to 2 seconds in time) are quite broader than the expected precision of hitting the ball at the middle of the racket at an exact timing (several centimeters in space and sub-second time scale). The normalized coordinates and timing of the ball are not discriminative when the player strokes the ball. We found that the use of auxiliary bounded parameters significantly improves the precision of control.

Data Augmentation The dribbling character was created using one minutes of dribbling motion data taken from the basketball data set. The character demonstrates a number of dribbling skills in response to the user's control. To evaluate its agility and responsiveness, we measured the average time to get to random targets 6 meters away from the character.

We performed a series of experiments by successively increasing the amount of data augmentation to demonstrate the effectiveness of our data augmentation method (see Figure 3.8). Control accuracy improves as the amount of synthetic data increases up to a certain threshold. After the threshold, additional data make no difference. The size of the required data augmentation depends on the content of the original mocap data. In our examples, we generated 4 hours of data, which is large enough, regardless of the data domain. It is just that we want to make the process automatic without the need of parameter tuning. We would also like to note that the relation between the learning speed and the size of training data is not linear. The relation is proportional when the training set is small, but the learning speed converges at a certain threshold, implying that excessive data expansion is less harmful than one might fear.

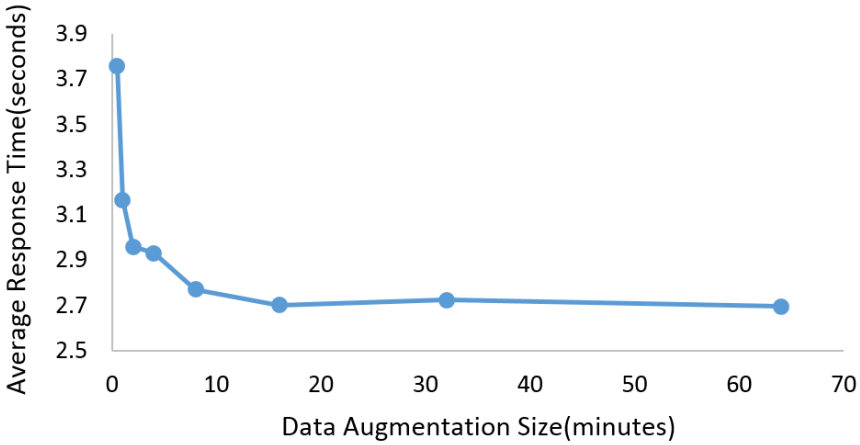

Figure 3.8 Agility by Data Augmentation.

Learning Locomotion from Small Datasets Our network model can be trained using a small amount of training data. A few short motion clips can learn a fully functional motion generator, though it is not as responsive as one may expect. The character can be more agile, responsive, and realistic with bigger and more diverse training data.

Teacher Forcing Teacher forcing is a technique that uses the ground truth of the past frame as input rather than the output of the network [52]. It allows the output to stay in the ground truth, and makes the learning faster and more stable in the training phase. We tested this technique, which helps to reduce the convergence time. However, the network thus obtained often exhibited artifacts with a complex example, such as basketball. It seems there is a trade-off between learning speed and robustness. All demos in this paper were learned without teacher forcing for high quality output.

3.7 Discussion

Recent video games provide rich details of characters' motion using large, high-quality motion databases. Many game developers have designed finite state machines and control mechanisms carefully to create interactively controllable game characters. Our model can be trained exploiting those readily-available motion databases, data structures, and control algorithms, which can generate episodic motion sequences randomly. The model learned from the synthesized training data can simulate the functionality of the existing systems while offering all the benefits of RNN-based online control, such as compact memory usage, computational efficiency at runtime, and flexibility in controller design.

The quality of the generated output mainly depends on the size and variability of the training data. Even though our data augmentation algorithm circumvents this issue to some degree, the diversity and responsiveness of character's motion are limited by the scope of the training data. Visual artifacts may be present if the animated character is controlled beyond its mobility and agility. We found that our model handles extreme control rather gracefully, avoiding undesirable visual artifacts such as jerky twitches and discontinuous jumps.

Recently, deep learning approaches have been quite successful in physics-based and data-driven animation. This trend does not imply that the conventional physics-based and data-driven methods will be obsolete. Existing animation methods will not be replaced by deep neural networks, which in fact supplement existing animation pipelines. As demonstrated in our work, many existing methods, such as motion graphs, motion grammars, and Laplacian motion editing, are essential components of learning deep neural networks. We found that deep learning can be an effective glue to put heterogeneous components together.

RNN vs Feedforward Neural Networks Previously, feedforward neural networks (FNNs) have been successfully applied to interactive character animation. RNNs have a lot of advantages over FNNs in modeling sequential, time-series data. FNN decides its action based on immediate observation at the current state, while RNN accounts for the trace of previous actions. In the work of Holden *et al.* [49], carefully designed user interfaces were used to provide structured control inputs, which include both past and future trajectories of biped locomotion. The structured input makes it possible to use FNN for sequential data synthesis. A latent phase variable also played an important role of aligning cyclic patterns of locomotion. As demonstrated by Zhang *et al.* [29] recently, the phase model can generalize to deal with a range of periodic and non-periodic actions using a Mixture of Experts model. They reported that a number of modes (or experts) should be chosen carefully to warrant stable convergence and avoid expert imbalance. Our RNN-based model requires neither structured inputs nor phase variables. We can design control inputs flexibly as an arbitrary combination of target position, target orientation, moving direction, moving velocity, and the timing and type of actions. Our model can learn many structurally-different actions all together without explicit representation of phases or modes, because the underlying dynamics and the connectivity of motions are learned in the recurrent network.

Chapter 4

Performance-driven Character Control with Parameter Adaptation

4.1 Overview

Control interfaces are an essential factor that influences user's experience and immersion in controlling virtual characters such as video games and virtual reality. The more information the control interface contains, the more user intent can be expressed, and more precise control can be achieved. From this point of view, performance-based control, which directly uses human behavior as a control interface is a very effective method. Recently, the development of motion sensors such as Microsoft's Kinect [53] and Asus Xtion [54], has made performance-based control easier to apply and used in various applications.

However, using the output of a motion sensor as it is is accompanied by several issues. The skeleton output by the motion sensor depends on the characteristics of the individual user. Mapping the skeletons of various users to a single virtual character requires a well-coordinated model. Also, since the motion sensor is used in a limited

space, the user can not walk or run over a long distance. This causes the controller to require extra information about the root movement of the virtual character. Another issue is that the controller learns the motion capture data of an expert actor, whereas the user is a novice of the performance. Even for beginners' awkward movements, a virtual character needs to move like a professional. This problem has previously been solved by using predefined action sets and physical simulations. These methods can handle only limited operations and require a lot of manual work to design the model according to the motion data.

Recently, a deep neural network has been successfully used as a tool for creating interactive character controllers with data-driven manner. Deep learning makes it possible to get the fast performance and low memory usage at runtime, and to utilize various motion databases without massive manual segmentation. However, previous approaches only deal with simple control parameters such as root trajectory or action type. The performance-based control approach using the high-dimension human pose as a control parameter has not been studied yet.

In this paper, we proposed a new performance-based controller based on deep learning using control parameter adaptation. The system is composed of the input parameter perturbation and parameter distribution prediction model. The input parameter perturbation model generates plausible noise to the control parameter at training time. This allows the control parameters for expert motions to be spread out over a broader range, allowing realistic movement to be output even for awkward performance of the user. The second model learns the distribution of control parameters from a given motion sequence using a new network. This serves to project a given input into an appropriate range at runtime if it deviates from the learned distribution of control parameters. This provides stability so that realistic motion can be gener-

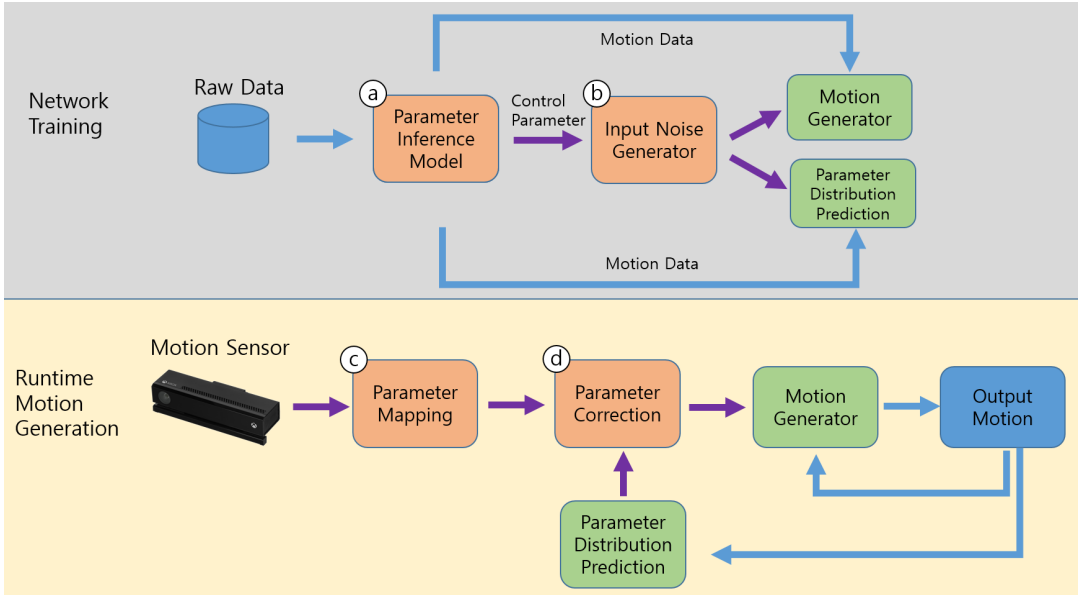

Figure 4.1 System Overview.

ated even from the immoderate input caused by the noise of the motion sensor or the character mapping error.

4.2 Performance-driven Character Control

Our system receives the stream of pose from the motion sensor and outputs the animation of the virtual character moving along the intention of the user. Our controller works in a data-driven manner, learning motions from the motion database and making the virtual character follow it. The goal of the controller can be divided into two, to follow the user's intention and to create a plausible motion. We used the RNN-based motion generator model described in chapter 3. The input of the network is the control parameter and the pose of the character in the current frame, and the output is the pose of the next frame.

$$m_{i+1} = RNN_g(c_i, m_i). \tag{4.1}$$

On the network learning process, it is necessary to inference the control parameter from the motion database. In order for the virtual character to follow the user's behavior, we used the motion after k frames as the control parameter. This teaches the controller so that the virtual character after the k frame has the same pose as the control parameter. We selectively use the position of the end-effector, joint orientation and velocity as control features.

$$\hat{c}_i = F(\hat{m}_{i+k}). \tag{4.2}$$

A hat above a symbol indicates the measurements in the training data. The control parameter at the runtime is calculated from the user's pose obtained from the motion sensor.

$$c_i = H(u_i). \tag{4.3}$$

where u_i is output of the motion sensor at frame i. Since the motion of the virtual character m_i and the motion of the user u_i have different data structures, it is important to carefully define the mapping function so that the two values are closely related. However, instead of adjusting the data mapping, we proposed a control parameter adaptation technique, which can effectively learn the controller even with simple control parameter mapping.

4.3 Input Parameter Perturbation

In the performance-based control, following a user's intention is an important issue, but excessively moving along a user's motions causes side effects. A novice user cannot

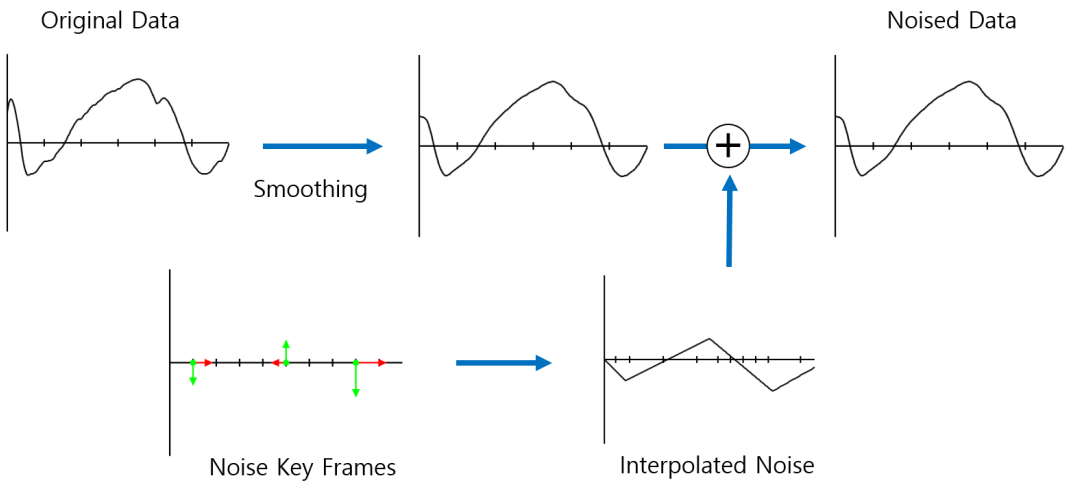

Figure 4.2 Input Parameter Perturbation.

precisely direct expert-level behavior. The user instructs the approximate motion and the controller should output elaborate and plausible motion based on the training data. However, the difference between the motion of the expert and the action taken by the novice user creates the difference of the control parameter at the training time and the runtime, and the controller can not output the correct motion when receiving the control parameter which was not covered at the training time. To address this problem, we introduced a method intentionally adding noise to the control parameters at training time.

We classified the difference between the training motion and the user's motion into three cases. The first case is the difficulty on following the detailed movements. For example, in a dance motion, it is difficult for an ordinary user to follow the action of quickly bending a hand or foot, or vibrate at a high speed. The user simply follows the trajectory of the arms and legs roughly. To solve this problem, we smoothed the control parameter to remove the detail information. This creates a mapping between a smooth control parameter and a detailed motion, allowing plausible motion to be

generated even if the user takes an incorrect motion at runtime. The smoothing process simply average the frames in the k time window.

$$s_i = average(\hat{c}_{i-k}, ..., \hat{c}_{i+k}). \tag{4.4}$$

where s_i is smoothed control parameter at training time. The second case is when the overall trajectory can not be followed. The second case is about the situation where the user can not follow the overall trajectory. When the user tries to instruct the kicking motion, the height of the foot taken by the user can be lower than the height of the motion-captured foot. We hope that even in such a case, a proper kicking action will occur, not a unskilled kicking motion. The last part is about the timing of the action. Because motion data is time series data, the speed of motion is also an important factor. Even if the user successfully follows the trajectory of the motion, the controller may be misunderstood if the speed of motion is different.

Due to the nature of the time series data, trajectory and timing error usually have continuous values. Based on this phenomenon, we added continuous noise to the control parameter values and timing. We first randomly sample the key frames from the control parameter sequence. Each key frame has random timing noise and data noise values. The control parameters of each frame have timing noise n_i and data noise d_i interpolating surrounding key frames. The final control parameter value is calculated by reflecting these noise values.

$$s'_i = curve(S, i + n_i) + d_i. \tag{4.5}$$

where S is sequence of the smoothed control parameter, and $curve(S, t)$ represents the value on the curve at time step t. The entire process is summarized on Figure 4.2.

4.4 Parameter Distribution Prediction Model

The skeleton of the motion data used in training time and the skeleton from the motion sensor at runtime may have different structures. The skeleton in the motion database is for one expert who performed motion capture. On the other hand, in the runtime, users can be a variety of people, man, woman, young and old. The general mapping algorithm can not completely cover these differences and causes an error on the control parameter between the training time and the runtime. Also, the output of the motion sensor is noisy, so some joints may not be able to obtain values due to occlusion of the body. To minimize these errors, we learned the distribution of control parameters from training data and corrected the user's control parameters at runtime.

4.4.1 Network Training

Parameter distribution prediction model(PDPM) takes current motion as input and outputs the distribution of the control parameter on the next frame. We model the probabilistic distribution of the control parameter using Gaussian Distribution Model. Note that the output of the PDPM is not a single control parameter but the mean and standard deviation of the control parameter.

$$p(c_{t+1}|m_t) = N(c_{t+1}|\mu_{t+1}, \sigma_{t+1}) \tag{4.6}$$

$$(\mu_{t+1}, \sigma_{t+1}) = PDPM(m_t) \tag{4.7}$$

where μ_t and σ_t is mean and standard deviation of the control parameter at time step t, respectively. PDPM uses only the motion of the current frame to predict the result and does not use the control parameter of the current frame. This allows the

model to predict the control parameter, even if the control parameter from the user is only partially presented or not at all.

The loss function is designed to maximize the probability of a control parameter for a given motion in the training data. Since the goal of the PDPM is compensating control parameters to the motion generator, it uses the same data used to learn the motion generator. This means that, the control parameter used for training is s' modified by input noise augmentation.

$$E = -\sum_{t} log\ p(s'_{t+1}|\hat{m}_t), \tag{4.8}$$

PDPM is learned using RNNs with a structure similar to motion generator. Our network model consists of multiple LSTM layers with encoder and decoder units (See Figure 4.3). This does not have an explicit recursive connection between the input and output of the network, but it is still possible to output considering the history because the states of the hidden layers are recursively updated.

4.4.2 Parameter Correction

At the runtime. we use PDPM to compensate the control parameters from the user. First, we obtain the distribution of the control parameter from the previous motion, and sample a random control parameter r_t from the distribution. Then, the control parameter c_t obtained from the motion sensor is interpolated with r_t as a ratio of k_r and used as an input for the motion generation network. The input parameter is constrained to not deviate by more than 2 standard deviations from the predicted distribution. If the value of some joints in c_t is missing, that dimension is replaced with the value of r_t.

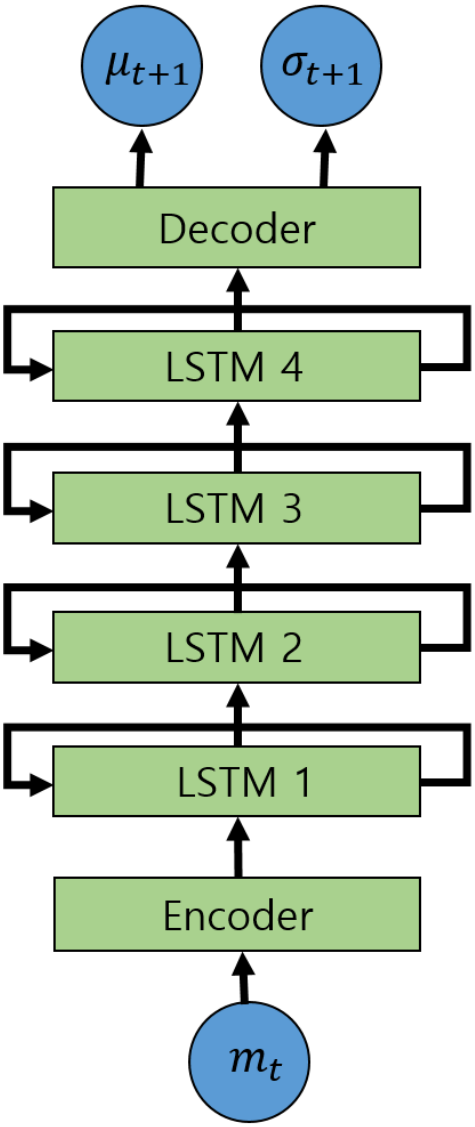

Figure 4.3 Network architecture of parameter distribution prediction model.

4.5 Experiments

We use a Microsoft Kinect (version 2) [53] to capture user performance. Skeletal information of the user was obtained using Microsoft's Kinect SDK. Since the SDK provides only joint position, we used the bone direction as the control parameter instead of the joint orientation. To utilize the motion sensor, the user can move within a limited space and can not reproduce realistic lower body motion. For this reason, We only used bone direction of upper body and velocity of hand position as control parameter. We takes control input in 30 Hz, and generated character motion with same frequency.

We implemented our algorithm in Python using TensorFlow [51] for the learning and evaluation of neural networks. The computing power of GPUs (NVIDIA GeForce GTX 1080) were utilized to accelerate neural network operations. We use multi-layer RNNs with four LSTM layers of size 512. The time step for truncated BPTT is 48. We used tanh activation function and Adam optimization algorithm. We inference control parameter from the character pose after 0.2 second. We utilized 12 minutes of basketball motion and 3 minutes of dance motion as training data. We built a motion graph for each data set, and generated 4 hours of augmented training data for each domain. The detailed training parameters are summarized in Table 4.5.

Parameter name	**Value**
Smoothing interval	0.3 second
Timing noise	-0.5 second to +0.5 second
Data noise	0.3 standard deviation
Parameter correction ratio	0.5

Table 4.1 Training parameters.

Figure 4.4 A snapshot of basketball demo.

Ablation Study. For qualitative evaluation, we compared the result with and without control parameter adaptation using basketball(Figure 4.4) and dance motion data(Figure 4.5). We used 6 frames as smoothing time interval, 0.3 standard deviation as data noise generation. Predicted control parameter is interpolated with user input with ratio 0.3. The result of baseline algorithm follows the novice user's awkward movements, but with control parameter adaptation the result shows plausible movements.

Plausibility evaluation. For quantitative evaluation, we measured plausibility of the result motion by changing smoothing time interval (see Figure 4.6). We assume the training motion data as ground truth, and considered similarity between the generated motion and the training data as a naturalness measure. We divide the generated motion into small clips with a length of 1 second and calculate the similarity by finding the motion most similar to the clip in the training data using dynamic time

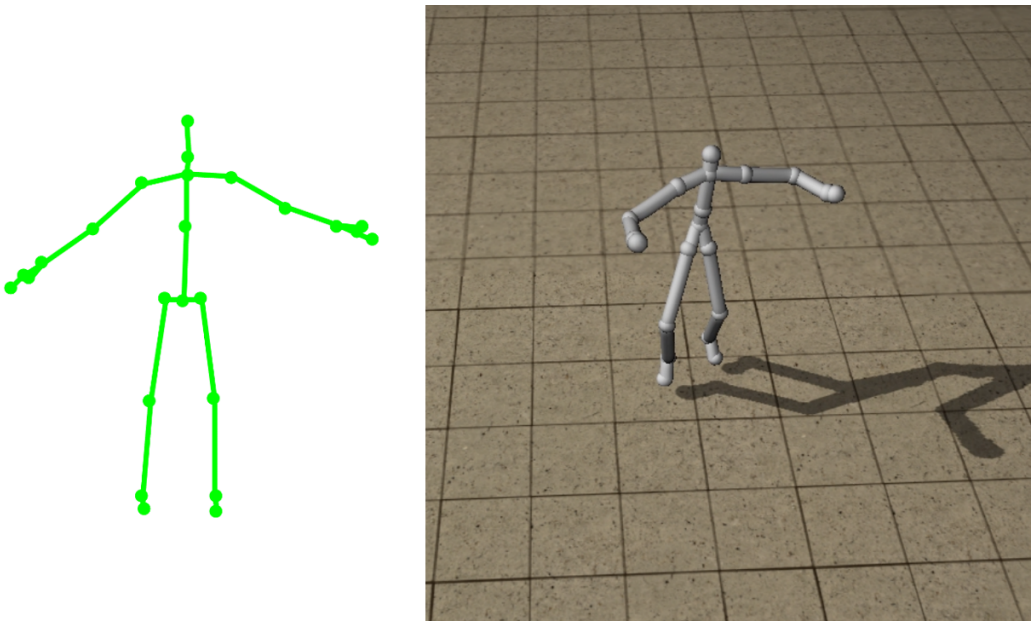

Figure 4.5 A snapshot of dance demo.

warping. Two minutes of user motion was used to generate motion for evaluation. The evaluation results show that the longer the smoothing interval, the more realistic the resulting motion.

4.6 Discussion

Performance-based control is an effective way to control a virtual character intuitively. Our control parameter adaptation algorithm reduces the gap between the user input and the control parameter of training data, so that the controller learns to generate a plausible motion from the awkward behavior of the novice user. We used a simple controller structure to verify the effectiveness of our algorithm. Our algorithm can be easily combined with other performance-based control methods, and there is potential for improvement in performance.

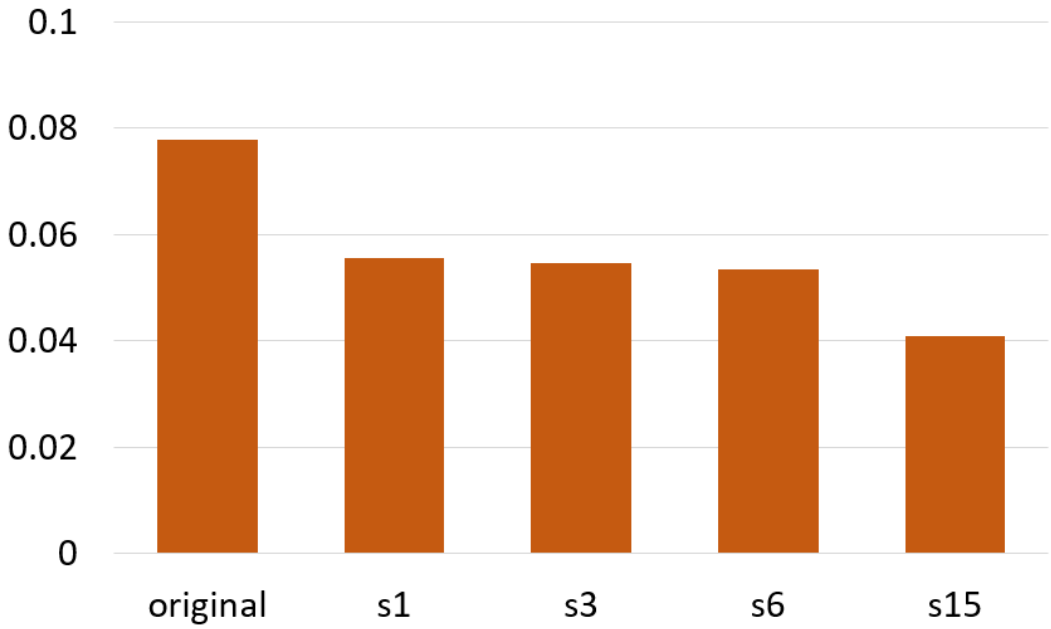

Figure 4.6 Plausibility evaluation. Each column shows dissimilarity of the result motion by various smoothing interval. The unit of dissimilarity is average standard deviation of each dimension.

Our model also has numerous limitations. The method of perturbing the control parameter is designed by intuition and it is not clear whether the modified parameter matches the actual user's performance. We are planning to verify the validity of our method by capturing the movement of user to follow learning motion data with motion sensor. The distribution of the control parameters is complex high-dimensional data, which is insufficient to be represented by a single gaussian distribution model. The introduction of a more complex distribution model, such as Principal Component Analysis, will help improve performance.

In data-driven character animation fields, controllability is a factor that is a trade-off relationship with plausibility [55]. With moderate amount of input parameter perturbation, our controller generate plausible movement following user's action. But, drastic perturbation degrades controllability and generates a plausible but random motion. It also has the effect of slowing convergence in the network training process. In this paper, we focused on increasing plausibility, but we wish to find a balance between plausibility and controllability or find ways to maintain controllability while improving plausibility as future works.

Chapter 5

Conclusion

Data-driven character animation is an effective way to reproduce realistic human motion. As the motion capture technology advances, various motion capture data are continuously accumulating. Nevertheless, there are still many issues in handling large amounts of motion data. The more motion data we have, the more effort we will need to label and segment the motion. The character controller should be able to contain more motions, require memory compactness and fast runtime performance. In this regard, we wanted to suggest a way to learn controllers that contain large amounts of data with less manual effort.

The thesis has proposed two approaches about character control. We first introduced a multi-objective control method using RNNs. A multi-objective model allows a single controller to learn various motions with different control parameters. The RNN-based model enables learning the structural pattern of the motion data, and inferring the intention of the actor allows the controller to be learned with minimal labeling. It also contains a data augmentation technique that allows robust controllers to be learned from small amounts of data. We also proposed a control parameter

adaptation method for performance-driven character control. Input parameter perturbation method enables the controller to output the expert's actions even from a novice user's unskilled actions. Parameter distribution prediction model allows the controller to work robustly even with skeleton differences between the user and the virtual character.

There are many exciting directions to explore, regard to the interactive control. One is to generate a complex scene interactively. The current model can produce scenes with a static basketball tactics board, but it is not enough to create a more complex scene or dynamically changing environment. We wish to interactively control scenes with intricate and dense interactions such as a fight scene. The other is learning a controller that can handle high-level manipulations. Using the proposed controller model, it is possible to learn the high-level controller which outputs the low-level control parameter to win the basketball game. The deep reinforcement learning model can be used here. Using RL, it is expected to be able to handle interactions with environments that were difficult to process with RNN, like a controller that moves to the destination avoiding complex obstacles.

Bibliography

[1] J. Lee, J. Chai, P. S. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 491–500, 2002.

[2] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 473–482, 2002.

[3] L. Kovar and M. Gleicher, "Automated extraction and parameterization of motions in large data sets," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 559–568, 2004.

[4] H. J. Shin and H. S. Oh, "Fat graphs: Constructing an interactive character with continuous controls," in *the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006, pp. 291–298.

[5] J. Min and J. Chai, "Motion graphs++: A compact generative model for semantic motion analysis and synthesis," *ACM Transactions on Graphics*, vol. 31, no. 6, 2012.

[6] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 522–531, 2004.

[7] S. Levine, J. M. Wang, A. Haraux, Z. Popović, and V. Koltun, "Continuous character control with low-dimensional embeddings," *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[8] K. Hyun, K. Lee, and J. Lee, "Motion grammars for character animation," *Computer Graphics Forum*, vol. 35, no. 2, pp. 103–113, 2016.

[9] G. Guerra-Filho and Y. Aloimonos, "The syntax of human actions and interactions," *Journal of Neurolinguistics*, vol. 25, no. 5, pp. 500–514, 2012.

[10] N. Dantam and M. Stilman, "The motion grammar: Analysis of a linguistic method for robot control," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 704–718, 2013.

[11] A. Safonova and J. K. Hodgins, "Construction and optimal search of interpolated motion graphs," *ACM Transactions on Graphics*, vol. 26, no. 3, 2007.

[12] J. Lee and K. H. Lee, "Precomputing avatar behavior from human motion data," in *the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004, pp. 79–87.

[13] A. Treuille, Y. Lee, and Z. Popović, "Near-optimal character animation with continuous control," *ACM Transactions on Graphics*, vol. 26, no. 3, 2007.

[14] V. Mnih and K. e. a. Kavukcuoglu, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] M. G. Choi, M. Kim, K. L. Hyun, and J. Lee, "Deformable motion: Squeezing into cluttered environments," *Computer Graphics Forum*, vol. 30, no. 2, pp. 445–453, 2011.

[16] S. Levine, Y. Lee, V. Koltun, and Z. Popović, "Space-time planning with parameterized locomotion controllers," *ACM Transactions on Graphics*, vol. 30, no. 3, 2011.

[17] K. H. Lee, M. G. Choi, and J. Lee, "Motion patches: Building blocks for virtual environments annotated with motion data," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 898–906, 2006.

[18] M. Kim, Y. Hwang, K. Hyun, and J. Lee, "Tiling motion patches," in *the 11th ACM SIGGRAPH/Eurographics Conference on Computer Animation*, 2012, pp. 117–126.

[19] H. P. Shum, T. Komura, M. Shiraishi, and S. Yamazaki, "Interaction patches for multi-character animation," *ACM Transactions on Graphics*, vol. 27, no. 5, 2008.

[20] J. Won, K. Lee, C. O'Sullivan, J. K. Hodgins, and J. Lee, "Generating and ranking diverse multi-character interactions," *ACM Transactions on Graphics*, vol. 33, no. 6, 2014.

[21] T. Shiratori and J. K. Hodgins, "Accelerometer-based user interfaces for the control of a physically simulated character," in *ACM Transactions on Graphics*, vol. 27, no. 5. ACM, 2008, p. 123.

[22] S. Ishigaki, T. White, V. B. Zordan, and C. K. Liu, "Performance-based control interface for character animation," in *ACM Transactions on Graphics*, vol. 28, no. 3. ACM, 2009, p. 61.

[23] J. Shotton, A. W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images." in *CVPR*, vol. 2, 2011, p. 3.

[24] M. Raptis, D. Kirovski, and H. Hoppe, "Real-time classification of dance gestures from skeleton animation," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation.* ACM, 2011, pp. 147–156.

[25] J. K. Tang, J. C. Chan, and H. Leung, "Interactive dancing game with real-time recognition of continuous dance moves from 3d human motion capture," in *Proceedings of the 5th international conference on ubiquitous information management and communication.* ACM, 2011, p. 50.

[26] Y. Lee and T. Kwon, "Performance-based biped control using a consumer depth camera," *Comput. Graph. Forum*, vol. 36, no. 2, pp. 387–395, May 2017. [Online]. Available: https://doi.org/10.1111/cgf.13134

[27] C. Mousas and C.-N. Anagnostopoulos, "Performance-driven hybrid full-body character control for navigation and interaction in virtual environments," *3D Research*, vol. 8, no. 2, p. 18, 2017.

[28] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Transactions on Graphics*, vol. 35, no. 4, 2016.

[29] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[30] L. Liu and J. K. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Transactions on Graphics*, vol. 36, no. 3, 2017.

[31] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[32] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills paper abstract author preprint paper video," *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[33] W. Yu, G. Turk, and C. K. Liu, "Learning symmetry and low-energy locomotion paper abstract author preprint paper video," *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[34] J. Won, J. Park, K. Kim, and J. Lee, "How to train your dragon: Example-guided control of flapping flight," *ACM Transactions on Graphics*, vol. 36, no. 6, 2017.

[35] J. Tan, Y. Gu, C. K. Liu, and G. Turk, "Learning bicycle stunts," *ACM Transactions on Graphics*, vol. 33, no. 4, 2014.

[36] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *the 28th International Conference on Machine Learning*, 2011, pp. 1017–1024.

[37] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *the 31st International Conference on Machine Learning*, 2014, pp. 1764–1772.

[38] F. A. Gers and E. Schmidhuber, "Lstm recurrent networks learn simple context-free and context-sensitive languages," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.

[39] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2625–2634.

[40] S. Suwajanakorn, S. M. Seitz, and I. Kemelmacher-Shlizerman, "Synthesizing obama: Learning lip sync from audio," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[41] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," in *Neural Information Processing Systems*, 2015, pp. 3132–3140.

[42] Y. Du, Y. Fu, and L. Wang, "Representation learning of temporal dynamics for skeleton-based action recognition," *IEEE Transactions on Image Processing*, vol. 25, no. 7, pp. 3010–3022, 2016.

[43] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *the IEEE International Conference on Computer Vision*, 2015, pp. 4346–4354.

[44] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5308–5317.

[45] P. Ghosh, J. Song, E. Aksan, and O. Hilliges, "Learning human motion models for long-term predictions," *arXiv preprint arXiv:1704.02827*, 2017.

[46] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 4674–4683.

[47] Y. Zhou, Z. Li, S. Xiao, C. He, Z. Huang, and H. Li, "Auto-conditioned recurrent networks for extended complex human motion synthesis," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r11Q2SlRW

[48] G. W. Taylor and G. E. Hinton, "Factored conditional restricted boltzmann machines for modeling motion style," in *the 26th Annual International Conference on Machine Learning*, 2009, pp. 1025–1032.

[49] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[50] M. Kim, K. Hyun, J. Kim, and J. Lee, "Synchronized multi-character motion editing," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[51] TensorFlow, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[52] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[53] Kinect, "Microsoft kinect motion capture system," 2015, software available from http://www.microsoft.com/en-us/kinectforwindows/.

[54] Xtion, "Assus xtion motion capture device," 2016, software available from http://www.asus.com/Multimedia/Xtion_PRO/.

[55] H. Van Welbergen, B. J. Van Basten, A. Egges, Z. M. Ruttkay, and M. H. Overmars, "Real time animation of virtual humans: a trade-off between naturalness and control," in *Computer Graphics Forum*, vol. 29, no. 8. Wiley Online Library, 2010, pp. 2530–2554.

요약

가상의 캐릭터를 조작하는 것은 그래픽스 분야의 중요한 문제중 하나이다. 캐릭터 컨트롤의 주요 목적은 캐릭터가 실제 사람과 같이 자연스럽게 움직이면서 사용자가 원하는 목적을 따라 행동하도록 하는 것이다. 자연스러운 동작을 재현하기 위해서는 실제 사람의 동작을 캡쳐한 모션 데이터가 활용된다. 다루고자 하는 동작이 많아질 수록 그만큼 더 많은 양의 모션 데이터를 필요로 하고, 각각의 동작에 해당하는 클립들을 분할하고 라벨링하는 작업을 필요로 한다. 이러한 분류 작업들은 사람이 직접 수행해야하기 때문에 이를 최소화 할 필요가 있다. 또한 각 동작을 조작하기 위한 매개변수들이 서로 다를 수 있기 때문에 여러 동작을 한번에 컨트롤하기 위한 모델이 필요하다. 실시간으로 사용자가 캐릭터를 조작하기 위해서는 런타임에 빠르게 계산이 가능해야하고 메모리 사용량이 적어야 한다.

본 학위논문에서는 심 신경망을 활용하여 조직화되지 않은 모션 데이터로부터 사용자 수작업을 최소화 하면서 실시간으로 조작 가능한 컨트롤러를 학습하는 기법을 제시하였다. 심 신경망을 활용하면 런타임에 학습 데이터를 가지고 있지 않아도 되기 때문에 메모리 사용량이 적고, 그래픽 카드 기반 가속 기법을 활용하여 실시간 연산이 가능하다. 우리는 순환신경망 모델을 활용하여 동작이 부드럽게 연결되면서 의미적으로 올바른 동작이 나오도록 하였다. 가상 캐릭터의 조작 방법으로는 두가지 기법을 제안하였다. 첫번째는 시공간 제약조건을 기반으로 다양한 동작들을 하나의 컨트롤러로 조작하는 것이다. 우리는 학습 모션 데이터로부터 캐릭터의 의도를 추론하여 각 동작의 액션 지점만 라벨링하면 자동으로 컨트롤러를 학습할 수 있는 모델을 제시하였다. 또한 기존의 캐릭터 애니메이션 기법을 활용한 데이터 증강 기법을 도입하여 적은 모션 데이터로부터도 안정적으로 캐릭터를 조작할 수 있도록 하였다. 두번째로는 사람의 행위를 기반으로 캐릭터를 조작하는 방법을 제안하였다. 사용자가 직접 몸을 움직여서 캐릭터를 조작하면

별도의 조작 방법을 익힐 필요 없이 직관적인 조작이 가능하다. 다만 가상의 캐릭터가 초보자인 사용자의 어설픈 동작을 그대로 따라가서 부자연스러운 결과가 나올 수 있다. 우리는 학습 데이터의 전문가 동작에 대한 조작 매개변수를 초보자의 입력과 유사한 형태로 변형하여 학습하여, 초보자의 입력에도 전문가와 같은 자연스러운 동작을 출력하는 컨트롤러 모델을 제안하였다. 제안한 모델의 효용성을 검증하기 위해 농구, 테니스, 댄스, 보행 등의 다양한 동작을 활용하여 결과를 비교하였다.

주요어: 컴퓨터 그래픽스, 데이터 기반 애니메이션, 캐릭터 제어, 실시간 제어, 행위 기반 제어, 기계 학습, 딥러닝, 순환신경망

학번: 2012-20823