Learning Time-Critical Responses for Interactive Character Control

KYUNGHO LEE, NCSOFT, South Korea
SEHEE MIN, Seoul National University, South Korea
SUNMIN LEE, Seoul National University, South Korea
JEHEE LEE, Seoul National University, South Korea

Creating agile and responsive characters from a collection of unorganized human motion has been an important problem of constructing interactive virtual environments. Recently, learning-based approaches have successfully been exploited to learn deep network policies for the control of interactive characters. The agility and responsiveness of deep network policies are influenced by many factors, such as the composition of training datasets, the architecture of network models, and learning algorithms that involve many threshold values, weights, and hyper-parameters. In this paper, we present a novel teacher-student framework to learn time-critically responsive policies, which guarantee the time-to-completion between user inputs and their associated responses regardless of the size and composition of the motion databases. We demonstrate the effectiveness of our approach with interactive characters that can respond to the user's control quickly while performing agile, highly dynamic movements.

CCS Concepts: • **Computing methodologies → Animation**; **Motion capture**; **Motion processing**.

Additional Key Words and Phrases: Interactive Character Control, Time-critical Response, Agility and Responsiveness, Recurrent Neural Network, Multi-Objective Control

ACM Reference Format:
Kyungho Lee, Sehee Min, Sunmin Lee, and Jehee Lee. 2021. Learning Time-Critical Responses for Interactive Character Control. *ACM Trans. Graph.* 40, 4, Article 147 (August 2021), 11 pages. https://doi.org/10.1145/3450626.3459826

Authors' addresses: Kyungho Lee, NCSOFT, 12, Daewangpangyo-ro 644ben-gil, Bundang-gu, Seongnam-si, 13494, South Korea, whcjs13@ncsoft.com; Sehee Min, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, South Korea, seiing@mrl.snu.ac.kr; Sunmin Lee, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, South Korea, sunmin.lee@mrl.snu.ac.kr; Jehee Lee, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, South Korea, jehee@mrl.snu.ac.kr.

1 INTRODUCTION

Creating agile and responsive characters is an essential goal of interactive character animation and its applications. Interactive characters are often created by acquiring a collection of motion capture data and manually crafting a control structure, such as finite state machines in game development, that allows transitioning between motion segments. Automated algorithms, such as motion graphs [Kovar et al. 2002; Lee et al. 2002] and motion matching [Clavet 2016], are also available to mitigate the manual effort. Recently, learning-based approaches have successfully been exploited to learn deep networks from human motion data [Holden et al. 2017; Lee et al. 2018]. The deep network acts as a motion generator and can control animated characters interactively at runtime.

The generative network model is often viewed as a black box that takes user control as input at any moment and generates the pose/motion of the character at the next time instance. The performance of the generative model can be evaluated in several aspects, such as motion quality, agility, responsiveness, and diversity. We wish that the model generates plausible human-like motions, allows the character to move quickly in response to user control, and equips the character with many action choices. Although these performance measures are of great importance in practical applications, learning a network model of desired quality level or improving the model in a certain aspect is a nontrivial task since deep network learning is a highly complicated process. Usually, the learning process consists of multiple steps, including data acquisition, pre-processing, augmentation, network learning, and post-processing, which involve many threshold values, weights, and hyper-parameters to tune. The performance of a network model is the result of complex interactions between training datasets and parameter tuning.

We are particularly interested in the agility and responsiveness of interactive characters. In this paper, a novel learning framework is presented to train a generative network policy based on a new concept, *time-critical responsiveness*. Our algorithm learns a time-critical policy that allows interactive characters to respond quickly to the user's control regardless of the size and composition of training datasets. The time-critically responsive network guarantees that the time duration between control input and its associated response is within a *critical response time*, which serves as the responsiveness level of the network policy. The key technical contribution of our work is as follows:

- *Time-critical responsiveness* is newly introduced in the context of interactive character control. The concept is popular in real-time systems. We adopted the concept to define the responsiveness of interactive characters.
- A novel *teacher-student* framework that first learns a teacher policy based on reinforcement learning and then a time-critical student policy by policy distillation. This framework simplifies the complexity of the problem to make it tractable.

2 RELATED WORK

Research on data-driven animation has explored a variety of approaches to utilizing motion capture data for motion synthesis and interactive character control. Many of the approaches share a common idea of allowing transitions between motion frames in the databases and eventually splicing them in a novel order. To do so, the motion graph explicitly maintains plausible transitions between motion frames in a directed graph [Kovar et al. 2002; Lee et al. 2002]. Any path traversing through the graph corresponds to a stream of the character's movements. Searching a graph path is a combinatorial planning problem, which has been addressed by using various methods, such as state-space search [Heck and Gleicher 2007; Min and Chai 2012; Safonova and Hodgins 2007], dynamic programming [Ren et al. 2010], and min-max search [Shum et al. 2012].

Given a motion graph, the agility and responsiveness of the character are closely related to the richness of potential transitions. The richer the transitions, the quicker the character can respond. Reitsma and Pollard [2007] evaluated the responsiveness of motion graphs by unrolling them into a virtual environment. Safonova and her collaborators [2010; 2009] studied ways to achieve good connectivity in motion graphs. McCann and Pollard [2007] created responsive game characters by establishing a mapping between the player's input and motion fragments. Ikemoto et al. [2007] computed the source-to-target blends of transitions during pre-processing and used the cached blends to make quick transitions at runtime. The perceptual effects of responsiveness are also evaluated in game environments [Jörg et al. 2012]. Ever since Lee and Lee [2004] introduced reinforcement learning into character animation, reinforcement learning has been frequently adopted to create interactive characters and their control policies [Lee et al. 2010; Levine et al. 2012; Treuille et al. 2007].

Motion matching is a variant of motion graphs aiming to maximize the responsiveness [Clavet 2016]. The motion matching algorithm does not construct a graph of transitions, but searches for the best transitions at runtime in response to the user's control. The difference between motion graphs and motion matching can be viewed from a trade-off between motion quality and responsiveness. Motion graphs pre-compute plausible transitions above a certain quality threshold, and runtime motion synthesis is limited within the pre-computed transitions. In other words, keeping the motion quality above the threshold is the main goal, and responsiveness is the secondary consideration. In contrast, the motion matching algorithm searches for the best transition among all possible transitions without limits. The motion quality should be assured by careful acquisition and pre-processing of the motion datasets, which should provide rich connectivity of good-looking transitions. There is a wealth of literature on data-driven animation dealing with multi-character interactions [Shum et al. 2008; Won et al. 2014], motion planning in static and dynamics environments [Choi et al. 2011; Kapadia et al. 2016; Lee et al. 2006; Levine et al. 2011; Lo and Zwicker 2008], syntactic structures [Hyun et al. 2016], and physical interactions [Arikan et al. 2005; Zordan et al. 2005].

Deep learning has brought revolutionary changes in data-driven animation since motion datasets can be effectively exploited for training deep networks. Holden et al. [2016] used an autoencoder and a feedforward neural network to produce motion sequences following a curve on the terrain. In their subsequent work, they learned a phase-functioned neural network to produce motions where the character adapts to uneven terrain [Holden et al. 2017]. Lee et al. [2018] used a recurrent neural network to learn various types of actions and user controls from unorganized human motion data. Recurrent models have also been used for motion prediction [Fragkiadaki et al. 2015; Martinez et al. 2017], inbetweening [Harvey et al. 2020], and synthesis [Henter et al. 2020]. Zhang et al. [2018] adopted a mixture of experts approach to deal with multiple modes in quadruped locomotion. This idea has further been explored to deal with character-scene interactions [Starke et al. 2019] and multi-modal phases [Starke et al. 2020]. Ling et al. [2020] combined variational autoencoder, phase-functioned neural network, and deep reinforcement learning to build a running controller for interactive characters. Motion graphs and motion matching have been used to provide datasets for training deep networks. For example, Lee et al. [2018] used motion graphs to generate training datasets and learn their recurrent networks. Holden et al. [2020] designed a network architecture that imitates the functionality of the motion matching algorithm.

Deep reinforcement learning (DRL) plays an important role in making a connection between data-driven animation and physics-based simulation. Peng et al. [2018] presented a DRL algorithm that learns a control policy to imitate a reference motion clip in physics simulation. Park et al. [2019] created physically-simulated characters that can be controlled interactively. To do so, they trained recurrent neural networks from human motion data to generate a stream of motion sequences in response to user input and utilized DRL to train a physics-based control policy to track the generated motion. Similarly, Bergamin et al. [2019] combined motion matching and DRL to create responsive simulated characters. Many control systems based on DRL have been explored to deal with anatomical bodies [Jiang et al. 2019; Lee et al. 2019], flexible object manipulation [Clegg et al. 2018], quadrupled locomotion [Luo et al. 2020; Peng et al. 2020], flapping flight [Won et al. 2017, 2018], swimming [Min et al. 2019], acrobatic movements [Liu and Hodgins 2017], basketball dribbling skills [Liu and Hodgins 2018], and large motion databases [Won et al. 2020].

3 TIME-CRITICAL RESPONSIVENESS

The *agility* of a dancer or athlete refers to the ability to move quickly and easily with a change of velocity or direction in response to a stimulus [Sheppard and Young 2006]. This is a different concept from *quickness*, which indicates a fast moving speed or an ability to change direction rapidly. The notion of *agility* entails perceptual recognition and decision-making components that lead to appropriate responses. It is also related to *responsiveness*, which refers to the duration of time between an input from the user and the associated response [Jörg et al. 2012]. In the context of our research, agility and responsiveness are interchangeable.

Interactive characters can be controlled to accelerate, decelerate, change their directions, switch poses, and perform actions, such as jump, run, crawl, punch, and dodge, through user interfaces. The user input, which is either continuous (e.g., joysticks) or discrete (e.g., push buttons), specifies a task for the character to perform. The agility of an interactive character (or its control policy) is defined by the expected response time to complete tasks given distribution of user inputs.

$$R(\pi) = \mathbb{E}_{s\sim P(s),u\sim P(u)} \left[\mathrm{TTC}_{\pi}(s,u)\right], \tag{1}$$

where TTC (Time-To-Completion) is the time duration to complete a task u starting from a state s. In our work, character animation is a discrete time-series system. The control policy π takes the character's current state and the user control as input and outputs the updated state for the next time instance.

Fast, delayless responses are of great importance in interactive applications. The easiest way to reduce the response time is to playback motion data faster than the data were originally captured. This brute-force speed-up of the whole motion datasets could result in the character's motion that looks hasty and unnatural. Alternative to minimizing the expected response time, we consider *Time-Critical Responsiveness (TCR)*, which measures the actual impact on perceived agility. TCR only considers cases where the completion time exceeds a user-specified threshold, which is called *critical response time*.

$$\mathrm{TCR}(\pi) = \mathbb{E}_{s\sim P(s),u\sim P(u)} \left[\mathrm{Truncated}_{\pi}(s,u)\right], \tag{2}$$

where the truncated time is

$$\mathrm{Truncated}_{\pi}(s,u) = \begin{cases} \mathrm{TTC}_{\pi}(s,u) - \tau(u), & \text{if } \mathrm{TTC}_{\pi}(s,u) > \tau(u), \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Since the time it takes to complete each task may differ, critical response time $\tau(u)$ is a function of user control u. We will discuss in the next section how to learn a control policy π while minimizing TCR.

Recursively applying the policy π starting from state s_0 with control u generates a motion sequence $M_{\pi}(s_0,u) = (s_0, s_1, \cdots, s_T)$, where T is the time-to-completion. The terminal state s_T satisfies completion criteria, which are task-dependent. We consider two types of tasks: maintenance tasks and achievement tasks. The completion criteria of a maintenance task require the character to maintain desired states (such as heading direction and walking speed) for a certain time duration. The achievement task is completed if the character reaches a terminal state (such as a key pose and a target location). Each task has its critical response time. The control policy π is time-critically responsive if it generates a motion sequence $M_{\pi}(s,u)$ for any given s and u to complete the task within its critical response time $\tau(u)$.

4 SYSTEM OVERVIEW

The goal of our research is to learn time-critically responsive policies from human motion data. There are very few assumptions about the motion databases, which may be unorganized and unsegmented. Rich transition-connectivity between frames is always preferred, but it is not strictly required. Usable policies can be learned even from sparsely-connected datasets. Our system can learn control policies regardless of the size of the databases, which may be as short as one minute and as long as a few hours in playtime.

The databases require minimal effort to annotate action types at motion frames. The annotations are used to command the interactive character. The databases include both periodic and aperiodic actions. Periodic actions are usually associated with maintenance tasks, while aperiodic actions are associated with achievement tasks. We annotated all motion frames of locomotion with appropriate maintenance task labels. For achievement tasks, we put labels on their keyframes including the beginning and ending of the action.

Our system can deal with various types of user control that include directional, positional, and velocity conditions. Each action type is associated with a tuple of user control parameters. For example, locomotion is controlled with direction and velocity parameters, while Jump-Over-Obstacle requires the position of the obstacle.

As demonstrated in previous studies, reinforcement learning (RL) is adept at learning control policies that allow the agent to perform desired tasks. However, we found that reinforcement learning alone is not sufficient to learn time-critical policies because of the structural complexity of motion data. Data-driven motion synthesis has exploited both combinatorial operations (e.g. splicing motion frames into a novel order) and continuous operations (e.g. smoothly deforming motion paths and time warping) to animate interactive characters. Incorporating both combinatorial and continuous operations into a unified framework is challenging. We present a teacher-student framework to circumvent this difficulty (see Figure 1). The teacher agent learns the optimal ways to achieve the tasks only allowing combinatorial operations. Once the teacher policy is learned, we train a student policy via policy distillation. Policy distillation is a process of extracting the policy of an RL agent and training a new network that performs the tasks at the same level of proficiency while being smaller and more efficient [Rusu et al. 2016]. We intervene in the policy distillation process through data generation and projection phases such that the student agent not only mimics the behavior of the teacher agent but improves on the policy in terms of response time and control accuracy.

The teacher and student policies have different goals and use different network architectures. The teacher policy has full access to the training datasets in both learning and inference phases. This privileged information enables the teacher policy to achieve high performance and better motion quality. We abandon the training

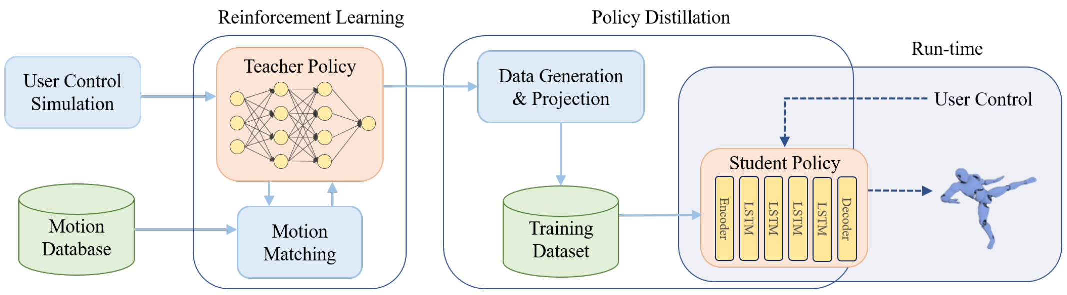

Fig. 1. System Overview

datasets after the learning phases for memory efficiency. Therefore, the student policy does not have such privileged access to the full datasets. The student policy uses a recurrent network model that produces future predictions based on an extended history of the states and thus can mimic complex control policies using brief state representation.

5 TEACHER LEARNING

We consider reinforcement learning that addresses discrete-time Markov decision problems. The RL agent takes the current state and the user control as input and outputs the next state while maximizing the expected cumulative rewards. The states correspond to the motion frames in the training datasets. The RL agent navigates through the training datasets similarly to how motion graphs and motion matching allow transitioning between motion frames and splicing them [Clavet 2016; Lee et al. 2002]. RL provides smarter, optimized policies to achieve the desired goals more efficiently. Our RL agent leverages both deep policy network and conventional, hard-coded decoder to implement this state transitioning process.

5.1 Reinforcement Learning Formulation

The user input is defined by a tuple $u = \{\hat{a}, \hat{t}, \hat{d}, \hat{p}\}$, where one-hot vector $\hat{a}$ represents the action type, $\hat{t}$ is the response time, $\hat{d}$ is the target direction, $\hat{p}$ is the target position. The target direction and the target position may or may not be specified depending on the action type. We simply put zero for unspecified values.

Let i be the index of motion frames in the training datasets and f_i be a motion frame that includes the description of a full-body pose. The state s is an enriched, feature-based description of f_i that includes the root position, root direction projected onto the ground, and the 3D positions of the end-effectors at the current frame i and three future frames at $i + 10$, $i + 20$ and $i + 30$. All positions are represented with respect to the body local coordinate system. Here, we assume that the motion data are captured at the rate of 30 frames per second. This rich, future-including description is preferred to learn from time-series data.

5.1.1 *State Transition.*

The output (action) of the policy network $\pi(s, u)$ is the change of the state vector Δs, which entails subsequent transitioning to a new state at the next time instance. The process of state transitioning is similar to motion matching. We search for the frame that best matches the state description $(s + \Delta s)$ from the training datasets. The new state s' is the state description of the best matching frame f_j. The user control u is also updated appropriately according to the change of the body coordinate system and the response time $\hat{t}$ is decreased by a fixed time step. Since frequent transitioning is harmful for motion quality, transitioning is allowed every N frames. In our experiments, we allowed transitioning every 5 frames. During this period of time, the agent proceeds to the next frame $j = i + 1$ in the datasets and motion frames are smoothly blended to exhibit seamless transitioning.

5.1.2 *Rewards.*

The reward includes two terms.

$$r = r_{\text{plausibility}} + r_{\text{task}} \tag{4}$$

The RL agent receives a plausibility reward when it makes a transition from state s to state s'.

$$r_{\text{plausibility}} = -w_{\text{plausibility}} \|f_j \ominus f_i\|^2, \tag{5}$$

where motion frames f_i and f_j, respectively, match the current and next state description s and s'. The symbol $\ominus$ represents the distance between two full-body poses in the databases. We compute the distance by the weighted sum of joint position differences and joint velocity differences. The plausibility reward encourages smooth transitions between frames. While the plausibility rewards are received continuously throughout the simulation, the task rewards are received sparsely only when a task is completed.

$$r_{\text{task}} = -w_d \|\hat{d} - d\|^2 - w_p \|\hat{p} - p\|^2, \tag{6}$$

where d and p, respectively, are the current direction and position of the agent. Here, r_{task} is maximized if the goal state $\hat{a}$ is achieved in the critical response time $\hat{t}$ and both target direction and target position are met at that time. In our examples, we always specify the action goal and the critical response time, but the target direction and target position are optional. If we do not have both, the reward is a constant, $r_{\text{task}} = 0$.

5.2 Pruning

The algorithm learns policy and value functions episodically. Given a random state s and a random user input u, policy π generates a motion sequence $M_\pi(s, u)$ to collect experience tuples. We generate many episodic motion sequences and update the policy and value networks using Proximial Policy Optimization [Schulman et al. 2017]. This on-policy approach can efficiently deal with maintenance tasks. However, the learning could be relatively slow for achievement tasks, which need to search for combinatorial paths through motion frames to reach the goal state. Gradient descent approaches for policy optimization are not particularly suitable for long-horizon combinatorial planning and search problems.

We dramatically improve the learning speed by precomputing the cumulative plausibility rewards for any state and response time. The precomputed plausibility rewards provide lower-bound of the cumulative rewards at any state and thus allow the learning algorithm to effectively prune unnecessary state exploration.

Let M be an episodic motion sequence of n frames, where the last frame of M meets the task goal a and its length $n < \tau$ is shorter than the critical response time. The cumulative reward of M is

$$V_a(M) = \gamma^n r^n_{\text{task}} + \sum_{i=0}^{n-1} \gamma^i r^i_{\text{plausibility}}, \tag{7}$$

where γ is a discount factor. We precompute and tabulate the second term in the equation. Specifically, we construct two tables $D_a(i,t)$ and $T_a(i,t)$. Assuming that the RL agent is in a state that matches the motion frame f_i and has a task a to achieve. The entry of the table $D_a(i,t)$ is the sum of discounted plausibility rewards assuming that the agent starting from frame f_i will make the most plausible moves to reach the goal state in time t. This motion is sub-optimal because it does not meet the target direction and target position when the goal state is achieved. $T_a(i,t)$ is a rigid transformation that represents the total translation and rotation of the agent while traversing the most plausible path. The construction of the tables is explained below.

Given the tables, the lower bound of the value can be derived.

$$V_a^{\text{lower}}(i,t) = r_{\text{task}}(T_a(i,t)) + D_a(i,t). \tag{8}$$

The task reward $r_{\text{task}}(T_a(i,t))$ assumes that the agent translates and rotates by $T_a(i,t)$ while it traverses following the most plausible path, which serves as an initial approximation of the optimal path. The policy learner does not need to explore any state of which expected value is below the lower bound and thus those state can be pruned in the learning process.

The pruning is implemented by modifying the state transitioning step in RL. Assume that the agent makes a transition from frame i to frame j since state $s(f_j)$ is the best match of state $s(f_i)$. The estimated plausibility reward can be derived from the Bellman equation.

$$V_a(i,j,t) = r_{\text{plausibility}}(i,j) + \gamma D_a(j,t-1). \tag{9}$$

This value should be higher than the lower bound. Otherwise, frame j is not worth exploring because it will never find a better path than the precomputed path. The modified matching process searches the best match subject to this pruning condition.

$$\begin{aligned} &\operatorname*{argmin}_j && \|s(f_j) - (s + \Delta s)\|^2, \\ &\text{subject to} && V_a(i,j,t) \geq V_a^{\text{lower}}(i,t). \end{aligned}$$

5.2.1 Precomputation. Table $D_a(i,t)$ is computed by dynamic programming, which sequentially fills in the cumulative plausibility rewards in the frame-time table. The recursive relation is

$$D_a(i,t) = \max_j \left(r_{\text{plausibility}}(i,j) + \gamma D_a(j,t-1) \right). \tag{10}$$

The size of the table is $N \times M$, where N is the number of motion frames in the database and M is the number of discretized time steps in $[0, \bar{\tau}]$. $D_a(i,0) = 0$ if the action label at frame i matches the user input. Otherwise, $D_a(i,0) = -\infty$. While updating the reward table during dynamic programming, we also update the transformation table synchronously to accumulate transformations along paths.

$$T_a(i,t) = T_{i\to j} \cdot T_a(j,t-1), \tag{11}$$

where $T_{i\to j}$ is the transformation to align the frames when the character jumps from frame i to j. Once the table is filled, the most plausible path to a goal state can be found by tracing backwards in the table. The dynamic programming is computationally feasible since the time horizon to explore is supposed to be short for agile and responsive characters.

6 POLICY DISTILLATION

Policy distillation is a process of supervised regression. The teacher policy generates a large collection of episodic motion sequences, which are used to train a student policy. The student policy replaces the teacher policy after learning.

The teacher policy is learned with a fixed critical response time τ_t for each task. The critical response time should be chosen carefully because there exists a trade-off between responsiveness and motion quality. If the critical response time is above a certain threshold, it does not affect the motion quality. However, if the critical response time is below the threshold, the agent has to make aggressive transitions between very different frames or may fail to meet the target direction/position within the time limit. The aggressive transitions could be a major cause of quality degradation. The threshold depends on the richness of transition-connectivity in the training datasets. If the datasets were captured in a carefully planned manner to allow rich transitions, the teacher policy is readily responsive and thus have a low critical response time. In that sense, τ_t is an intrinsic parameter that can be derived from the datasets.

The student policy is not an identical copy of the teacher policy, but the student policy achieves better responsiveness through policy distillation. Responsiveness can be improved by making motion data flexible. Motion sequences in the training datasets allow for spatial deformation and time warping to improve control accuracy and responsiveness. For the student policy, critical response time is a controllable parameter. The student policy can have a continuous spectrum of responsiveness ranging from agile to sluggish by adjusting the parameter τ at run-time, where the parameter range is $\tau \in [\tau_s, \tau_t]$. Unlike the upper-bound τ_t, the lower-bound τ_s is specified by the user to determine how agile the interactive character can get.

6.1 Data Generation and Projection

Given an initial state s and a user control u, the teacher policy produces an episodic motion sequence $M_0(s,u)$ with a subsequent state s' at the end of the motion. The next episode begins with state s'. Repeating this process with a random distribution of user control produces a batch of episodes $\mathcal{M} = \{M_i\}$, which forms a long continuous sequence of motion frames suitable for training recurrent networks.

All motion sequences generated by the teacher policy are time-critically response, so shorter than τ_t. We want to generate more variations to deal with controllable parameter τ for the student policy. To do so, we draw random samples from the range $[\tau_s, \tau_t]$ and timewarp the motion sequence if it is longer than the random sample. In this way, we can collect many tuples of a time limit and a motion sequence that performs a desired take within the time limit.

The motion sequence may not perfectly satisfy the direction and/or position conditions at the end frame. The main reason of this inaccuracy is the training datasets that cannot provide sufficient variations for continuous control. The inaccuracy can be fixed by smoothly deforming the motion sequence to enforce the direction/position conditions.

We employ the Laplacian motion editing [Kim et al. 2009] to perform both spatial deformation and time warping. Given spatial and temporal constraints, the motion sequence undergoes deformation by adjusting both its spatial trajectory and timeline smoothly. The rigidity can be specified per-frame basis in the Laplacian formulation. The rigidity coefficients determine how easily the spatial trajectory deforms and the timeline warps. It is desirable that key actions remain unchanged and the subsidiary actions between key actions are deformed to improve responsiveness. Labeling higher rigidity coefficients on key actions achieves the desired result through non-linear spatial deformation and non-linear time warping.

6.2 Student Learning

The student policy takes the current state s_t and user control u_t as input and outputs the next state s_{t+1}.

$$s_{t+1} = \pi(s_t, u_t) \tag{12}$$

We adopted a Recurrent Neural Network (RNN) model since it is simple, easy-to-implement, and adept at dealing with time-series data. Specifically, the network model consists of stacked LSTM layers sandwiched between encoder and decoder layers, similar to the multi-objective control model [Lee et al. 2018].

The state of the student policy is represented as a tuple $s = \{r_t, r_d, j_p, j_r, c, a\}$, where $r_t \in \mathbb{R}^2$ and $r_d \in \mathbb{R}^1$ are the position and direction, respectively, of the skeleton root on the two-dimensional plane. j_p and j_r, respectively, is an aggregated vector of joint positions and orientations. The orientation of a joint is represented by two column vectors of the rotation matrix. The third vector can be omitted because it is the cross product of the other vectors. c is a vector of binary flags that indicate the body-ground contacts at end-effectors. a is a one-hot vector indicating the activation of an action.

The training dataset $\mathcal{M}$ provides a rich collection of experience tuples (s_t, u_t, s_{t+1}). The student policy can be learned from the experience tuples via supervised learning. The loss function includes three terms:

$$E = E_{\text{motion}} + E_{\text{contact}} + E_{\text{rigid}}. \tag{13}$$

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

$$E_{\text{motion}} = \sum_t \|s_t - \hat{s}_t\|^2, \tag{14}$$

where $\hat{s}_{t+1}$ and s_{t+1}, respectively, are the observed state from an experience tuple and the corresponding network output. The contact term E_{contact} penalizes foot sliding.

$$E_{\text{contact}} = \sum_t \sum_{k \in \text{Feet}} c_t^k c_{t+1}^k \|h_t^k - h_{t+1}^k\|^2, \tag{15}$$

where k is the joint index and c_t^k is a binary flag that indicates if the joint k is in contact with the ground plane at time t. h_t^k is the joint position in the reference coordinate system. The rigid term E_{rigid} favors that the lengths of the body links are preserved.

$$E_{\text{rigid}} = \sum_t \sum_{(k,m) \in \text{Links}} (\|h_t^k - h_t^m\| - l_{km})^2, \tag{16}$$

where two adjacent joints k and m are connected by a rigid link and l_{km} is its length.

7 EXPERIMENTS

We used TensorFlow [TensorFlow 2015] for deep network operations and Unreal Engine [Epic Games 2019] for characterization, interactive control, and rendering. All computations were run on a single PC (Intel i7-9700 CPU spec and NVIDIA GeForce RTX 2080Ti GPU). The Adam optimizer was used in all learning processes. The policy and value networks in RL consist of four fully-connected layers of 512 ReLU nodes. The learning rate of policy and value networks, respectively, are 10^{-4} and 10^{-3}. The discount factor γ is 0.99 and GAE parameter λ is 0.95. The clip range of PPO is 0.2. The batch size is 256. The student network consists of four LSTM layers with 512 tanh nodes and two fully-connected encoder/decoder layers. The learning rate is 10^{-4} and the batch size is 128. The time step for truncated BPTT is 48.

The teacher learning takes 12 to 24 hours in computation, while the student learning takes 4 to 8 hours. Highly energetic actions tend to require more computation to learn. The unoptimized network inference module for run-time simulation takes 10 milliseconds per frame with a single character. The network inference module optimized with OpenVINO toolkit exhibits better performance such that 100 characters in a scene can be animated simultaneously at the rate of 10 milliseconds per frame.

We utilized a collection of motion databases either available freely on the web or commercially on the Unreal Engine asset store (see Table 1). The motion frames are labeled to identify action types. Specifically, we identified four actions (Walk, Jog, Run, and Stop) in the locomotion data. For each aperiodic action, we labeled a keyframe (e.g., the highest point of a jump and the moment of impact of a kick) that serves as a reference of completing the action. We also labeled a time interval around the keyframe, which matches the beginning and ending of the action. The foot-ground contacts are identified automatically. Foot contact times are automatically labelled by considering the distance and velocity of the heel and toe joints relative to the ground plane.

7.1 Locomotion

The locomotion dataset consists of about 40,000 motion frames (22 minutes playtime) including walking, jogging, running, spiral turns with various radii and rapid u-turns. The character is controlled using a joystick, which specifies the moving direction. The user

Table 1. Motion databases

Name	Motion Length	Training Time	Action Types
Locomotion	21.5 minutes	2 hours	Stop, Walk, Jog, Run
Obstacles	1 minute	2 hours	Jog, Two-Hand Jump, One-Hand Jump, No-Hand jump, BackFlip
Martial Arts	4 minutes	3 hours	Idle, Punch, Kick, Jump Kick, Backspin Kick, Jump Backspin Kick, Side Flip
Super-Agile	2 minutes	2.5 hours	Stop, Run, Dash, Dash Left, Dash Right, Slide, Front Jump, Back Jump, High Jump
Horse	5.5 minutes	3 hours	Walk, Trot, Canter, Stop, Walk Backward, Roar, Jump

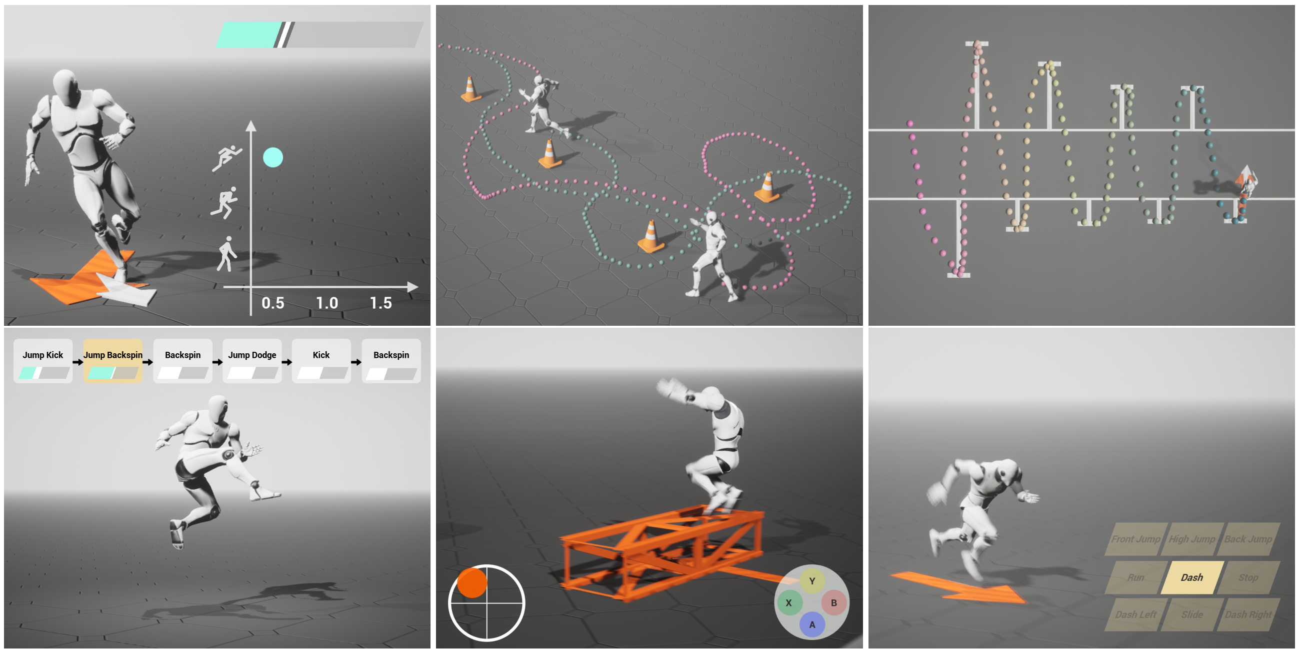

Fig. 2. Image snapshots of examples. (Top, Left to Right) Quickness and responsiveness in locomotion. Slalom. Shuttle run. (Bottom, Left to Right) Martial arts, Jumping over obstacles. Super-agile.

can choose from three locomotion types (walk, jog, run) to change the moving speed. The critical response time τ is a controllable parameter in the user input, which allows us to modulate the responsiveness interactively at runtime. The examples can be best viewed in the supplementary video (also see Figure 2).

7.1.1 Distribution of Time-To-Completion. For any given locomotion type and turning angle, the character is supposed to complete the task by τ. Figure 3 shows the plots of the completion time demonstrating how our network policy responds to direction changes. Let θ_i be a turning angle in the user's control and $t_i = \mathrm{TTC}_\pi(\theta_i)$ be the time-to-completion (TTC) when it is controlled by our locomotion (student) policy π. We draw a collection of n tuples $\{\theta_i, t_i\}$ by running the simulation repeatedly with a random distribution of θ_i. The kernel density estimation (KDE) is used for better visualization of the distribution. The expected TTC is

$$\mathbb{E}_{\mathrm{TTC}}(\theta) = \frac{\sum_i t_i \cdot K(\frac{\theta - \theta_i}{h})}{\sum_i K(\frac{\theta-\theta_i}{h})}, \tag{17}$$

where K is a Gaussian kernel function with bandwidth $h = 1.06\sigma n^{-\frac{1}{5}}$, as suggested by Silverman's rule of thumb [Silverman 1986], and σ is the standard deviation of the samples. As expected, the larger turning angle requires more time to complete the task. As the critical response time decreases, the agility improves gradually mainly at large turning angles. There remain outliers above the critical response time in the plot. The outliers are removed slowly as the learning progresses. Since the learning has a long tail, we early stopped the learning when the learning curve plateaus and the outliers are almost unperceivable in simulation.

7.1.2 Comparison to Motion Matching. We conducted experiments to compare our learned policy with the motion matching algorithm [Clavet 2016]. Figure 4 shows a series of the probability density functions and kernel density estimations of the completion time while varying τ from 0.333 seconds to 1.0 second. Our locomotion policy with a large response time behaves similarly to the motion matching algorithm, which poses a trade-off between motion quality and responsiveness. The weight w favors feature matching (responsiveness) over frame continuity and smoothness (motion quality).

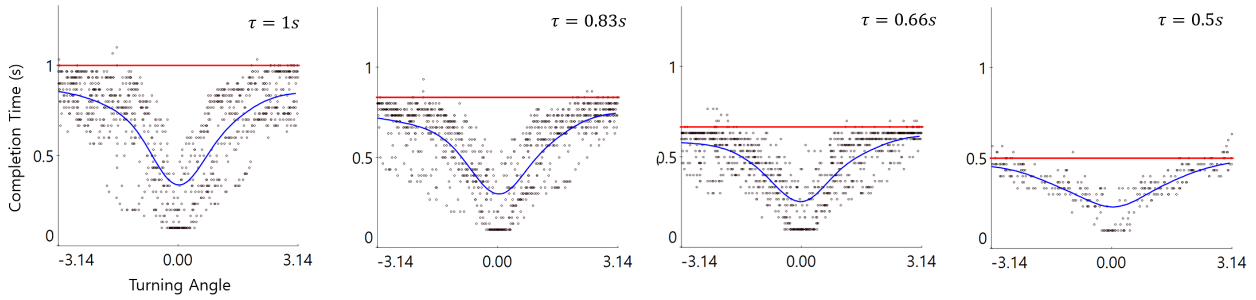

Fig. 3. The completion time plots for locomotion. The critical response time varies from 0.5 seconds to 1.0 seconds. The blue curve shows the expected completion time computed using KDE.

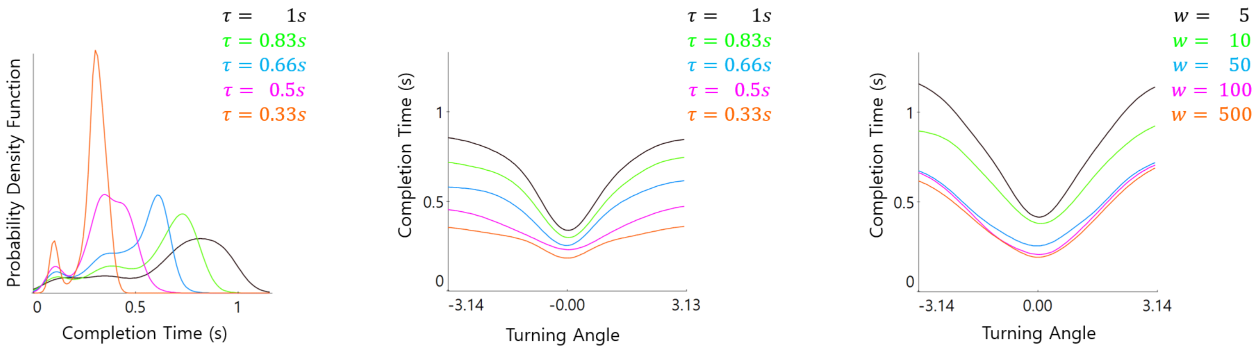

Fig. 4. The expected completion time for locomotion. (Left) Probability density of our locomotion policy, (Middle) KDE plots for our locomotion policy, (Right) KDE plots for motion matching.

Larger weights allow for more aggressive transitions. In the figure, the KDE plot of our learned policy with $\tau = 1.0$ looks similar to the KDE plot of motion matching with $w = 5$. Unlike our network policy, the responsiveness of motion matching plateaus at $w = 50$ and does not improve thereafter. The limit is inherent as long as the timings in the motion databases are fixed factors that cannot be modified or generalized. Our learning algorithm overcomes this limitations by enabling spatial and temporal generalization for the student policy and thus achieves better responsiveness and better motion quality over motion matching.

7.1.3 *Slalom and Shuttle Run.*

We have created several examples to demonstrate how the level of responsiveness affects the interactability and controllability in interactive applications. The *Shuttle Run* example shows the character running back and forth between two lines making rapid U-turns while the system increases its responsiveness continuously from $\tau = 1.5$ to 0.5. The moving trajectory shows large overshoots with large τ. The U-turn trajectory makes smaller overshoots as the agility improves with smaller τ. In the *Slalom* example, the user controls the character using a joystick to do a slalom race. The agile character ($\tau = 0.5$) clears the slalom easily along a smooth trajectory, while the normal character ($\tau = 1.0$) makes abrupt movements to pass through the cones and sometimes bumps into them.

7.2 Martial Arts

The *Martial Arts* dataset contains many aperiodic actions such as kick, punch, jump and dodge. The goal of the character is to perform a chain of martial arts movements in the queue as rapidly as possible. The critical response time can vary depending on which action follows which action. We defined the normal agility by estimating $\tau(a, a')$ between a pair of consecutive actions from the motion datasets. $\tau(a, a')$ is the maximum of all shortest paths between frames with action label a and frames with action label a'. This setting guarantees that the teacher policy is time-critically responsive.

The critical response time for the student policy varies in the range of 40% to 100% of $\tau(a, a')$ between any a and a'. Figure 5 shows the min, average, max time to complete four actions (Backspin-Kick, Jump-Kick, Jump-Backspin-Kick, and Side-Flip) when it starts from running. The student network successfully learned the generalization of the teacher policy for the range of different time limits.

7.3 Keyframe Animation

Artist-created motion data are often stylistic, exaggerated, and available as a packet of short motion clips with a manually-crafted FSM (finite state machine). *Super-Agile* is one of those manually created datasets, which are quite different from motion capture data. This

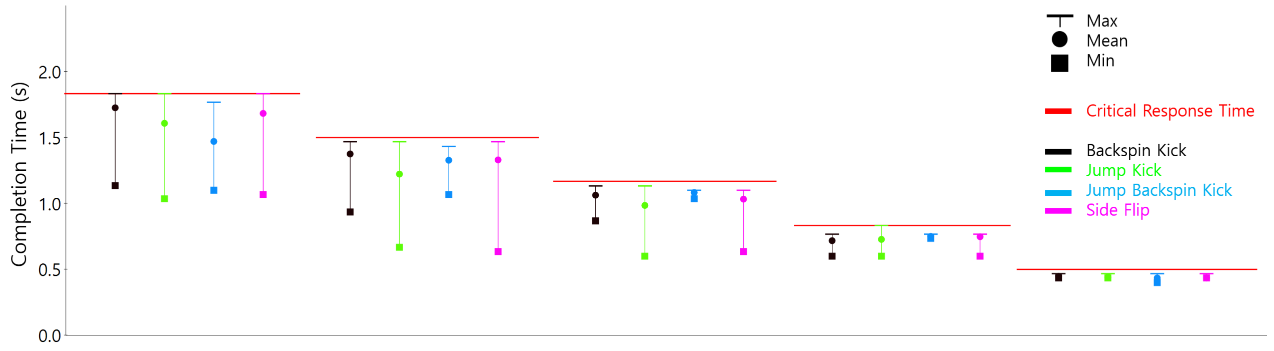

Fig. 5. The completion time of martial arts movements.

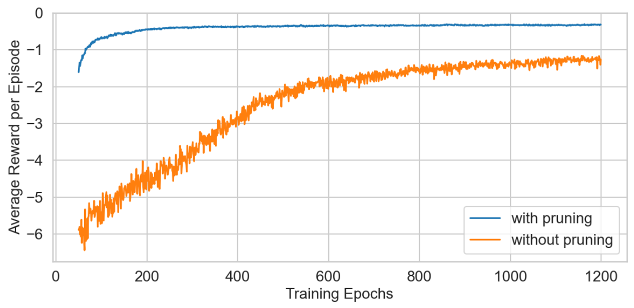

Fig. 6. Teacher learning curves for the obstacles dataset.

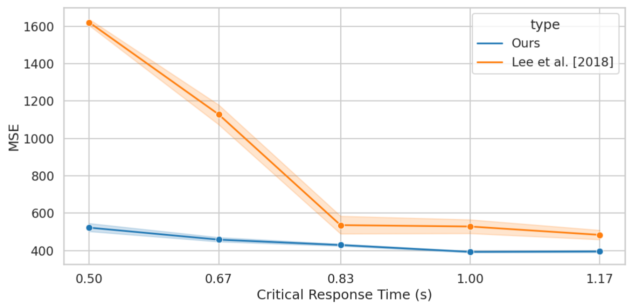

Fig. 7. Motion quality comparison with Lee et al. [2018]. The MSE value represents the difference between the generated motion and the original motion data.

example demonstrates that our network can also deal with artist-created motion data. It is also possible to integrate artist-created and motion capture data seamlessly to allow time-critical transitions between them.

7.4 Position Control

Precise position control is required to interact with the virtual environment. The character trained with the *Obstacles* dataset can run around and jump over obstacles with one or two hands. The user controls the character using a joystick and commands the character to jump when it is near an obstacle. Since the network policy can do position control, the user does not need to place the character in the exact location before triggering the command.

We compared the performance of teacher learning with and without pruning. Figure 6 shows the learning curves for the *Obstacle* dataset. Pruning improves the learning performance dramatically.

7.5 Horse

Learning quadruped locomotion is the same as learning biped locomotion. The horse character has four limbs and a tail with total 42 ball-and-socket joints. The motion capture dataset includes about 10,000 motion frames (5.5 minute playtime). The dataset includes various gaits (walk, trot, and canter) and many actions such as roar, jump, rapid stop, and walk backwards. Given a desired speed, the horse character can automatically choose appropriate gaits to match the walking speed.

7.6 Comparison

We compared our method against an RNN-based multi-objective control (MOC) model [Lee et al. 2018] trained on the jump motion dataset. The MOC method was also trained to perform jump actions within the same critical response time τ. As the critical response time decreases, the quality of output motions degrades gradually. The accompanied video shows that our method better handles short response times than the MOC method (see the supplementary video for qualitative comparison). We measured the quality of the generated jump motion quantitatively in comparison to the original motion data (Figure 7). The MSE (Mean Square Error) measures the difference in the joint position after applying dynamic time warping (DTW). Our method shows a good balance between motion quality and responsiveness even with critical response times shorter than physically-plausible ranges.

7.7 Ablation on Policy Distillation

The expressive power of the teacher policy is the same as motion graphs and motion matching in the sense that the output is generated by traversing the motion databases, except that RL makes smarter decisions based on long-horizon planning capabilities at every moment. Policy distillation can significantly improve the expressive power of the policy network through spatial deformation and time warping. We experimented with the effect of policy distillation by changing the critical response time on the direction control example. Teacher policy works well for long response times but is not satisfied with short critical response time beyond the range expressed as the training dataset. Policy distillation allows short critical response time regardless of the amount of training dataset. The richness of training data can reduce the loss of motion quality to achieve fast response speed.

8 DISCUSSION

We presented a learning-based approach to create agile and responsive characters from unorganized human motion databases. Our teacher-student framework achieves high responsiveness, high control accuracy, and a wide range of action choices while minimizing the potential risk of quality degradation and manual effort for labeling the motion databases. Our approach shares all the benefits of learning-based methods, such as memory efficiency and fast runtime performance. Any post-processing (such as inverse kinematics to avoid foot sliding) is not needed since the direct outputs from the network model are accurate and readily usable.

Separation of a teacher policy and a student policy simplifies the complexity of the problem and improves the performance significantly. It is very challenging to learn all the spatial and temporal conditions and controls in a single network model. However, the problem is manageable if it is separated into two subsequent learning problems. We take advantage of the planning capabilities of reinforcement learning to address the combinatorial aspects of the problem, while the student model uses the expressive power of recurrent networks to address continuous control in the spatial and temporal domains.

Even though DRL and the recurrent network model served well for our system design, our framework is agnostic to the network architecture. Any other network architecture, such as attention networks and Transformer [Vaswani et al. 2017], can replace the recurrent model as long as the conditional discrete time-series model can be trained via supervised regression.

There are a number of limitations and opportunities for future work. In this work, we focused on improving the agility and responsiveness of control policies. There are other high-level properties we might want to evaluate from control policies and improve. For example, we would like to generate a natural variety of actions even when the user input is static (diversity). We would like to exploit the motion databases comprehensively without dead spots (coverage). We sometimes want to have each individual action to be clearly articulated without blurring at the beginning and ending of the action (articulation). We believe that algorithms similar to ours can be applied to address those high-level properties.

ACKNOWLEDGMENTS

This work was supported by NCSOFT AI Center under Project Number NCSOFT2020-0061-K. The 2nd, 3rd, and 4th authors were partly supported by Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) (2017-0-00878, SW Starlab).

REFERENCES

Okan Arikan, David A. Forsyth, and James F. O'Brien. 2005. Pushing People Around. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005*. 56–66.
Kevin Bergamin, Simon Claver, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-Driven Responsive Control of Physics-Based Characters. *ACM Transactions on Graphics* 38, 6, Article 206 (2019).
Myung Geol Choi, Manmyung Kim, Kyunglyul Hyun, and Jehee Lee. 2011. Deformable Motion: Squeezing into Cluttered Environments. *Computer Graphics Forum* 30, 2 (2011).
Simon Clavet. 2016. Motion Matching and The Road to Next-Gen Animation. In *GDC 2016*.
Alexaander Clegg, Wenhao Yu, Jie Tan, Karen C. Liu, and Greg Turk. 2018. Learning To Dress: Synthesizing Human Dressing Motion via Deep Reinforcement Learning. *ACM Transactions on Graphics* 37, 6 (2018).
Epic Games. 2019. *Unreal Engine*. https://www.unrealengine.com
Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*. 4346–4354.
Felix Harvey, Mike Yurick, Christopher Pal, and Derek Nowrouzezahrai. 2020. Robust Motion In-Betweening. *ACM Transactions on Graphics* 39, 4 (2020).
Rachel Heck and Michael Gleicher. 2007. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*. 129–136.
Gustav Eje Henter, Simon Alexanderson, and Jonas Beskow. 2020. MoGlow: Probabilistic and Controllable Motion Synthesis Using Normalising Flows. *ACM Trans. Graph.* 39, 6, Article 236 (2020).
Daniel Holden, Oussama Kanoun, Maksym Perepichika, and Tiberiu Popa. 2020. Learned Motion Matching. *ACM Transactions on Graphics* 39, 4 (2020).
Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics* 36, 4, Article 42 (2017).
Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Transactions on Graphics* 35, 4, Article 138 (2016).
Kyunglyul Hyun, Kyungho Lee, and Jehee Lee. 2016. Motion grammars for character animation. *Computer Graphics Forum* 35, 2 (2016), 103–113.
Leslie Ikemoto, Okan Arikan, and David Forsyth. 2007. Quick transitions with cached multi-way blends. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*. 145–151.
Yifeng Jiang, Tom Wouwe, Friedl De Groote, and Karen C. Liu. 2019. Synthesis of Biologically Realistic Human Motion Using Joint Torque Actuation. *ACM Transactions on Graphics* 38, 4 (2019).
Sophie Jörg, Aline Normoyle, and Alla Safonova. 2012. How responsiveness affects players' perception in digital games. In *Proceedings of the ACM Symposium on Applied Perception*. 33–38.
Mubbasir Kapadia, Xu Xianghao, Maurizio Nitti, Marcelo Kallmann, Stelian Coros, Robert Sumner, and Markus Gross. 2016. Precision: precomputing environment semantics for contact-rich character animation. In *Proceedings of the 2016 symposium on Interactive 3D graphics and games*. 29–37.
Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. 2009. Synchronized Multi-character Motion Editing. *ACM Transactions on Graphics* 28, 3, Article 79 (2009).
Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. *ACM Transactions on Graphics* 21, 3 (2002), 473–482.
Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. 2002. Interactive Control of Avatars Animated with Human Motion Data. *ACM Transactions on Graphics* 21, 3 (2002), 491–500.
Jehee Lee and Kang Hoon Lee. 2004. Precomputing Avatar Behavior from Human Motion Data. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 79–87.
Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-Objective Control. *ACM Transactions on Graphics* 37, 6, Article 180 (2018).
Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. 2006. Motion Patches: Building blocks for virtual environments annotated with motion data. *ACM Transactions on Graphics* 25, 3 (2006), 898–906.
Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. 2019. Scalable Muscle-actuated Human Simulation and Control. *ACM Transactions on Graphics* 38, 4, Article 73 (2019).
Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010. Motion Fields for Interactive Character Locomotion. *ACM Transactions on Graphics* 29, 6, Article 138 (2010).
Sergey Levine, Yongjoon Lee, Vladren Koltun, and Zoran Popović. 2011. Space-time planning with parameterized locomotion controllers. *ACM Transactions on Graphics*

30, 3 (2011).

Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous Character Control with Low-Dimensional Embeddings. *ACM Transactions on Graphics* 31, 4 (2012).

Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel van de Panne. 2020. Character Controllers Using Motion VAEs. *ACM Transactions on Graphics* 39, 4 (2020).

Libin Liu and Jessica Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Transactions on Graphics* 36, 3 (2017).

Libin Liu and Jessica Hodgins. 2018. Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning. *ACM Transactions on Graphics* 37, 4 (2018).

Wan-Yen Lo and Matthias Zwicker. 2008. Real-time planning for parameterized human motion. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2008*. 29–38.

Ying-Sheng Luo, Jonathan Hans Soeseno, Trista Pei-Chun Chen, and Wei-Chao Chen. 2020. CARL: Controllable Agent With Reinforcement Learning for Quadruped Locomotion. *ACM Transactions on Graphics* 39, 4 (2020).

Julieta Martinez, Michael J Black, and Javier Romero. 2017. On human motion prediction using recurrent neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 4674–4683.

James McCann and Nancy Pollard. 2007. Responsive characters from motion fragments. In *ACM Transactions on Graphics*, Vol. 26. 6.

Jianyuan Min and Jinxiang Chai. 2012. Motion graphs++ a compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics* 31, 6 (2012), 1–12.

Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. 2019. SoftCon: Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators. *ACM Transactions on Graphics* 38, 6, Article 208 (2019).

Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. 2019. Learning predict-and-simulate policies from unorganized human motion data. *ACM Transactions on Graphics* 38, 6, Article 205 (2019).

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics* 37, 4, Article 143 (2018).

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. 2020. Learning Agile Robotic Locomotion Skills by Imitating Animals. *arXiv preprint* (2020).

Paul SA Reitsma and Nancy S Pollard. 2007. Evaluating motion graphs for character animation. *ACM Transactions on Graphics* 26, 4 (2007), 18.

Cheng Ren, Liming Zhao, and Alla Safonova. 2010. Human motion synthesis with optimization-based graphs. In *Computer Graphics Forum*, Vol. 29. 545–554.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. 2016. Policy Distillation. arXiv:1511.06295 [cs.LG]

Alla Safonova and Jessica K Hodgins. 2007. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics* 26, 3, Article 106 (2007).

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]

Jeremy M Sheppard and Warren B Young. 2006. Agility literature review: Classifications, training and testing. *Journal of sports sciences* 24, 9 (2006), 919–932.

Hubert Shum, Taku Komura, and Shuntaro Yamazaki. 2012. Simulating Multiple Character Interactions with Collaborative and Adversarial Goals. *IEEE Transactions on Visualization and Computer Graphics* 18 (2012), 741–52.

Hubert PH Shum, Taku Komura, Masashi Shiraishi, and Shuntaro Yamazaki. 2008. Interaction patches for multi-character animation. *ACM Transactions on Graphics* 27, 5, Article 114 (2008).

Bernard W Silverman. 1986. *Density estimation for statistics and data analysis*. Vol. 26. CRC press.

Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. 2019. Neural State Machine for Character-Scene Interactions. *ACM Transactions on Graphics* 38, 6, Article 178 (2019).

Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Transactions on Graphics* 39, 4 (2020).

TensorFlow. 2015. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. http://tensorflow.org/

Adrien Treuille, Yongjoon Lee, and Zoran Popović. 2007. Near-optimal character animation with continuous control. *ACM Transactions on Graphics* 26, 3, Article 7 (2007).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

Jungdam Won, Deepa Gopinath, and Jessica Hodgins. 2020. A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters. *ACM Transactions on Graphics* 39, 4 (2020).

Jungdam Won, Kyungho Lee, Carol O'Sullivan, Jessica K Hodgins, and Jehee Lee. 2014. Generating and ranking diverse multi-character interactions. *ACM Transactions on Graphics* 33, 6, Article 219 (2014).

Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. 2017. How to Train Your Dragon: Example-guided Control of Flapping Flight. *ACM Transactions on Graphics* 36, 6, Article 198 (2017).

Jungdam Won, Jungnam Park, and Jehee Lee. 2018. Aerobatics control of flying creatures via self-regulated learning. *ACM Transactions on Graphics* 37, 6, Article 181 (2018).

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics* 37, 4, Article 145 (2018).

Liming Zhao and Alla Safonova. 2009. Achieving good connectivity in motion graphs. *Graphical Models* 71, 4 (2009), 139–152.

Victor Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. 2005. Dynamic response for motion capture animation. *ACM Transactions on Graphics* 24 (2005), 697–701.