Neural State Machine for Character-Scene Interactions

SEBASTIAN STARKE†, University of Edinburgh, United Kingdom
HE ZHANG†, University of Edinburgh
TAKU KOMURA, University of Edinburgh
JUN SAITO, Adobe Research, USA

Fig. 1. A selection of results using our method to generate scene interaction behaviors.

We propose *Neural State Machine*, a novel data-driven framework to guide characters to achieve goal-driven actions with precise scene interactions. Even a seemingly simple task such as sitting on a chair is notoriously hard to model with supervised learning. This difficulty is because such a task involves complex planning with periodic and non-periodic motions reacting to the scene geometry to precisely position and orient the character. Our proposed deep auto-regressive framework enables modeling of multi-modal scene interaction behaviors purely from data. Given high-level instructions such as the goal location and the action to be launched there, our system computes a series of movements and transitions to reach the goal in the desired state. To allow characters to adapt to a wide range of geometry such as different shapes of furniture and obstacles, we incorporate an efficient data augmentation scheme to randomly switch the 3D geometry while maintaining the context of the original motion. To increase the precision to reach the goal during runtime, we introduce a control scheme that combines egocentric inference and goal-centric inference. We demonstrate the versatility of our model with various scene interaction tasks such as sitting on a chair, avoiding obstacles, opening and entering through a door, and picking and carrying objects generated in real-time just from a single model.

CCS Concepts: • **Computing methodologies** → **Motion capture**; *Neural networks.*

Additional Key Words and Phrases: neural networks, locomotion, human motion, character animation, character control, deep learning

ACM Reference Format:
Sebastian Starke†, He Zhang†, Taku Komura, and Jun Saito. 2019. Neural State Machine for Character-Scene Interactions. *ACM Trans. Graph.* 38, 6, Article 178 (November 2019), 14 pages. https://doi.org/10.1145/3355089.3356505

1 INTRODUCTION

A character's relation to its environment largely defines the fidelity of its motion. Let us take a task of sitting on a chair as an example. The character must accurately find a path going around obstacles such as a table or the target chair itself. The character must also plan to precisely position and orient itself from the target chair. Other everyday tasks such as opening doors and carrying objects involve very different motions but share the common challenge of precise control based on the scene understanding.

High-quality motion generation using deep supervised learning has shown success in the limited domain around locomotion [Holden et al. 2017, 2016]. Extending it to support various everyday tasks involving scene interactions remains as an unsolved problem because the model needs to launch a series of complex actions and transitions to reach the goal with high-level scene understanding, while accurately adapting to the wide variation of 3D geometries.

In this paper, we propose *Neural State Machine* to model a wide range of periodic and aperiodic movements, including locomotion, sitting, standing, lifting and collision avoidance. Our neural architecture internally acquires a state machine for character control by learning different motions and transitions from various tasks. By designing a control signal with the phase of motion encoded into high-level action labels and goal locations, our neural network learns to generate precise production-quality animations from high-level commands to achieve various complex behaviors, such as walking towards a chair and sitting, opening a door before moving out from a room and carrying boxes from or under a desk. Instead of enforcing

† Sebastian Starke and He Zhang are joint first authors.

Authors' addresses: Sebastian Starke†, University of Edinburgh, 10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom, sebastian.starke@ed.ac.uk; He Zhang†, he.zhang@ed.ac.uk, University of Edinburgh; Taku Komura, tkomura@ed.ac.uk, University of Edinburgh; Jun Saito, jsaito@adobe.com, Adobe Research, 801 N 34th St, Seattle, WA, 98103, USA.

0730-0301/2019/11-ART178 $15.00
https://doi.org/10.1145/3355089.3356505

a fixed phase function to factorize the network weights [Holden et al. 2017], which only suits cyclic movements such as locomotion, our system learns such a factorization from the motion capture data in an end-to-end fashion.

To further increase the precision of goal-driven movements, where existing systems suffer from error accumulation during inference, we propose a bi-directional scheme where inference from the egocentric point of view as well as from the goal-centric point of view synthesizes the motion. The results are blended and fed back into the network to produce smooth movements where the character is guided to the goal with high precision.

To overcome issues with heightmap-based representations as used previously, we introduce volume representations for understanding the environment so our system can learn movements adapting to objects with concavity, such as passing through a hole in a wall or sliding onto a chair sideways to avoid a desk. To further generalize the scene adaptation, we augment the training data by randomly switching the geometry in each frame, while maintaining the context of the motion and interaction. Our data augmentation scheme allows the system to learn the interactions with a wide variation of geometry efficiently without increasing the data size.

We demonstrate examples such as a character moving around in a room with obstacles, interacting with furniture and moving objects at different locations. The system runs in real-time and can be applied for character control in computer games, virtual reality, and augmented reality.

The contribution of our system can be summarized as follows:

- a control signal and an autoregressive architecture design enabling automatic transitions of vastly different high-level action states to reach the given goal state, while maintaining motion quality in real-time,
- a bi-directional control framework that mixes inference from an egocentric point of view and that from a goal-centric point of view for improving the precision of goal-driven tasks,
- a volumetric representation for guiding the characters to interact with objects and environments with concavities,
- an efficient data augmentation scheme to learn the interaction with a wide range of geometry while preserving the context and without increasing the data size and
- a comprehensive analysis of our method and comparison with state-of-the-art approaches.

The rest of the paper is summarized as follows. In Section 2, we provide a brief survey of the related work. Next in Section 3, we describe the architecture of our system, and then how we make use of it for goal-driven motion synthesis in Section 4. In Section 5, we describe how we prepare our training data. We show the experimental results in Section 6 and evaluate the results in Section 7. We discuss the limitations of our system in Section 8 and then conclude the paper in Section 9.

2 RELATED WORK

We first review techniques that are developed to animate close interactions between characters and the environment. We next review machine learning techniques that can potentially be applied for the problem of close character-environment interaction.

2.1 Animating Close Interactions

Interaction Synthesis from Motion Capture Sequences. Various kinematics based techniques that make use of motion capture data are proposed for animating close interactions between the body and the environment. Lee et al. [2002] present results of animating characters doing jungle gym activities such as climbing up ladders, hanging on monkey bars etc. under the Motion Graph framework. This setup only allows the characters to move in the same environment as when the motion was captured, which limits the scalability of the method. One approach that allows animating the characters in novel environments is to use the patch-based approach: Motion Patches [Lee et al. 2006] use building blocks where the character interacts with the environment or with other characters as units and assemble them to generate large scale scenes. Results such as characters playing on the slide, sitting and working on desks and interacting with other characters are presented. The use of patch-based approaches are extended to close interaction of characters [Kim et al. 2012; Shum et al. 2008] and synthesis of movies that follow some scenario [Won et al. 2014]. Agrawal and van de Panne [2016] animate close interactions with the environment by aligning the foot step patterns of the motions in the database.

These approaches are template-based approaches: a selected motion clip is inserted into the scene and edited according to the scenario. Template-based approaches are simple but do not produce a continuous motion space that is needed for constructing a wider variation of movements.

To cope with the problem with template-based approaches, techniques to apply kernel-based approaches are explored for synthesizing novel character movements [Mukai and Kuriyama 2005; Rose et al. 1998; Rose III et al. 2001; Wang et al. 2008]. As multiple motion clips are interpolated to synthesize a novel motion, a wide variation of smooth and natural motion can be constructed by adjusting the blending weights. The issue with kernel-based approaches is in their scalability and the amount of preprocessing needed: when the data size is very large, the nearest neighbour search can be time consuming and become the bottleneck of the approach. Also large amount of data must be preserved in the memory. Deep learning architectures described in Section 2.2 try to overcome this issue.

Motion Synthesis by Exploration. Another approach to synthesizing novel interactions with objects and environments is through exploration. Methods based on probabilistic roadmaps [Choi et al. 2003], rapidly exploring random trees [Choi et al. 2011], A* search [Safonova and Hodgins 2007] reinforcement learning [Lo et al. 2012; Lo and Zwicker 2008] and Maximum A Posteriori (MAP) estimation [Min and Chai 2012], local optimization [Mordatch et al. 2012] and stochastic optimization. [Liu et al. 2010; Ye and Liu 2012; Zhao et al. 2013] are applied for exploring novel movements for interactions.

Choi et al. [2003] generate probabilistic road maps in the environment based on footstep patterns and use them to generate characters that move in the environment. Motions to walk through narrow areas with side stepping can be generated under their framework. Similarly, Choi et al. [2011] apply online RRT expansion for navigating characters through narrow environments - whether the character can pass through a narrow environment or not is checked

by evaluating the collision of the body's sweep surface and the environment. Lo et al. [2012; 2008] apply reinforcement learning for navigating characters to move through doors [Lo and Zwicker 2008] and plan the motion to reach a goal in environments with obstacles [Lo et al. 2012]. These approaches mostly replay existing motion capture data and there is no discovery of movements or synthesis of unseen novel movements.

Safonova and Hodgins [2007] use a parametric motion graph structure where the two motions can be interpolated to produce a novel motion. The selection of the motions as well as the blending weights are computed by A* search. Similarly, Min et al. [2012] propose a Motion Graph++ structure where motions of the same type are blended by functional PCA. They optimize the blending weights by MAP estimation. Our work is similar to these approaches in the sense that we also have annotations for each motion type and the system automatically interpolates between them for reaching the goal. By making use of a neural network as a representation, our system can learn from a large amount of high dimensional data, such as volumetric data of objects and environments and human motion capture sequences, resulting in a wider range of movements. Also, as we adopt a supervised learning scheme, the motion can be synthesized without optimization steps, making the character responsive and controllable in real-time.

Synthesis of novel movements can be done by local or stochastic optimization. Mordatch et al. [2012] propose a local optimization scheme that produces close interactions from scratch based on a contact-invariant optimization. The method can synthesize novel movements such as characters passing luggage from one to another just from an objective function that specifies the placement transition. Ye et al. [2012] produce finger motions for manipulating objects by randomly sampling the contact points of the fingers and the objects. Liu et al. [2010] successfully track the movements of motion capture data in physically-based environments by sampling postures around the captured motion data. Zhao et al. [2013] generate grasping movements by stochastic optimization. Local optimization is fast but is unstable especially when the object to interact with is convex. Sampling based approaches are more stable but are slow for real-time character control. Our focus in this research is thus learning from human motion capture data in a supervised fashion for real-time character control.

2.2 Deep Learning Techniques for Synthesizing Close Interactions

The usage of neural networks for synthesizing character movements is rapidly increasing, due to their large scalability and their ability to produce novel motion data. Such approaches can be divided into supervised learning approaches and deep reinforcement learning approaches. Finally, we discuss the controllability of the outputs produced by the deep networks.

Motion Synthesis by Deep Supervised Learning. Methods to train controllers from motion capture data based on LSTMs, temporal convolution, Phase-Functioned Neural Networks and its variants are explored in the machine learning, computer vision and computer graphics community.

LSTMs are considered suitable for time-series models and are applied for reproducing locomotion [Fragkiadaki et al. 2015], kung fu motion [Li et al. 2017] and sports activities [Lee et al. 2018] as well as for motion retargeting [Villegas et al. 2018] and interpolating keyframe postures [Harvey and Pal 2018]. LSTMs often suffer from converging to average poses due to the ambiguity in the future. This is more obvious especially for simple repeating problems without control signals. Li et al. [2017] overcome this by applying probabilistic teacher forcing. For problems such as character control, there is less ambiguity in the future when providing control signals. Lee et al. [2018] construct a four layer LSTM model for animating characters to conduct locomotion and to play basketball and tennis. One issue with recurrent neural network structures such as LSTMs is in the difficulty of training them. Bai et al. [2018] present the difficulty of training LSTMs for sequence modelling. We also find an issue in training LSTMs in our experiments (see Section 7 for our problem), which could be due to the large state space of the internal memory.

Holden et al. [2017; 2016] instead use temporal convolutions [Holden et al. 2016] and a specialized architecture called Phase-Functioned Neural Networks [Holden et al. 2017] for producing sharp movements. Zhang et al.[2018] propose a method based on the mixture of experts [Jacobs et al. 1991] to construct a real-time character controller for quadruped characters. The advantage of the concept of mixture of experts is in its scalability: a wide range of character movements can be constructed under the same framework as experts specialized for different movements can be trained. We further explore this direction for animating humanoid characters conducting close interactions in this paper.

Motion Synthesis by Deep Reinforcement Learning. Deep Reinforcement Learning (DRL) is widely applied for controlling characters in a physically-based domain, for motions such as locomotion [Peng et al. 2016], flapping [Won et al. 2017], avoiding obstacles [Heess et al. 2017], acrobatic movements [Peng et al. 2018a,b] and dressing [Clegg et al. 2018]. When applying DRL for controlling human characters, movements generated by simple rewards based on the velocity and balance often appear unnatural [Heess et al. 2017]. Peng et al. [2018a; 2017; 2018b]. cope with this problem by providing reference motion capture data [Peng et al. 2018a, 2017] or video data [Peng et al. 2018b].

When applying these approaches for low energy, ordinary daily life motion such as simply standing up from a chair and moving to another chair, idling, defining the reward function could be an issue. These types of motions are not governed by simple, single objective and a reward that results in natural low energy movements is needed. Yu et al. [2018] propose a reward that encourages symmetry to produce natural locomotion. For physically-based character control, further exploration of the reward is needed for the wide variety of low energy movements in ordinary daily life that involve close interactions and multiple contacts.

3 NEURAL STATE MACHINE

The Neural State Machine (Fig. 2) consists of a *Motion Prediction Network* and a *Gating Network*. The Motion Prediction Network is the main component responsible for the auto-regression of the

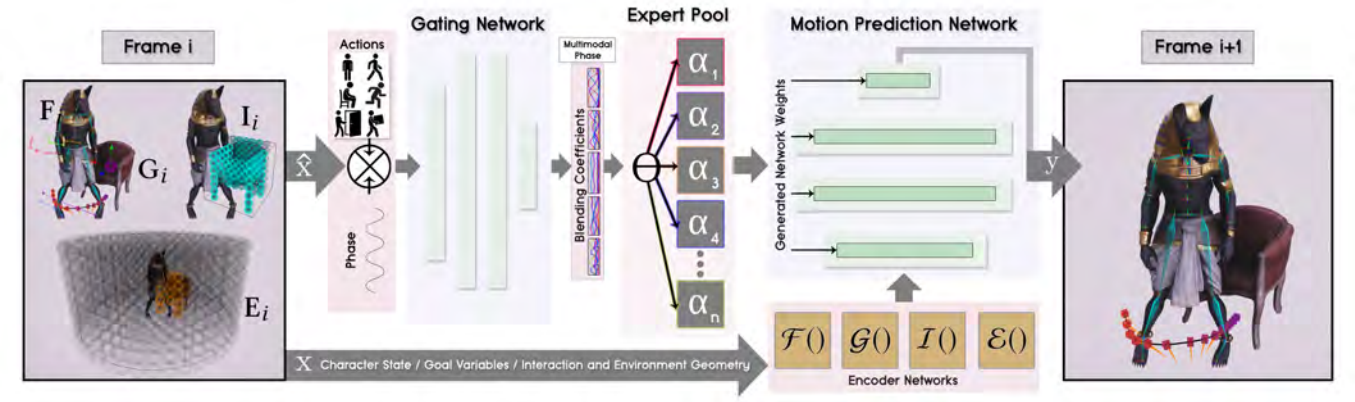

Fig. 2. The architecture of our system composed of the gating network and the motion prediction network. The gating network takes as input a subset of parameters of the current state and the goal action vector to output the expert blending coefficients which is then used to generate the motion prediction network. The motion prediction network takes as input the posture, trajectory control variables and the goal parameters from the previous frame, and predicts those variables for the current frame.

character-related information. The Gating Network decides the blending coefficients of expert weights based on the given goal and the phase of the motion to dynamically generate the Motion Prediction Network. The intuition behind this design is to encode different modes of motion in drastically different tasks separately in multiple experts to avoid blurry motions and adapt to different tasks. We further modulate the expert blending weights using a learned cyclic function to ensure the state will move forward in time.

3.1 Motion Prediction Network

The Motion Prediction Network (see Fig. 2, right) is the main component of our system where it computes the pose of the character in the current frame, given the pose in the previous frame, the geometry of the surrounding environment, and the high-level instructions including the goal location and the action state.

The Motion Prediction Network is composed of two modules where the first module is the **encoder module** that receives the components of the character state in the previous frame and encodes them individually using simple three-layer networks. The second module is the **prediction module** that receives the output of the encoder module and predicts the state of the character in the current frame. The prediction module is a three-layer network whose weights are dynamically blended from a group of expert weights with coefficients from the Gating Network.

Inputs:

The inputs of the Motion Prediction Network $\mathbf{X}_i$ at frame i consist of four components, namely the Frame Input $\mathbf{F}_i$, Goal Input $\mathbf{G}_i$, Interaction Geometry Input $\mathbf{I}_i$ and Environment Geometry Input $\mathbf{E}_i$, such $\mathbf{X}_i = \{\mathbf{F}_i, \mathbf{G}_i, \mathbf{I}_i, \mathbf{E}_i\}$ (see Fig. 2, left). Each of these items are described below.

- *Frame Input* $\mathbf{F}_i = \left\{\mathbf{j}^p_{i-1}, \mathbf{j}^r_{i-1}, \mathbf{j}^v_{i-1}, \mathbf{t}^p_{i-1}, \mathbf{t}^d_{i-1}, \mathbf{t}^a_{i-1}\right\}$, is composed of the following items:
  - *Character Pose in Frame* $i-1$*:* $\mathbf{j}^p_{i-1} \in \mathbb{R}^{3j}$, $\mathbf{j}^r_{i-1} \in \mathbb{R}^{6j}$ $\mathbf{j}^v_{i-1} \in \mathbb{R}^{3j}$ are joint positions, rotations and velocities relative to the root coordinate in frame $i-1$, and j denotes the 23 joints of the character.
  - *Past/Future Root Trajectory:* $\mathbf{t}^p_{i-1} \in \mathbb{R}^{2t}$, $\mathbf{t}^d_{i-1} \in \mathbb{R}^{2t}$ are the root coordinate positions and forward directions of the t ($t = 13$) sampled trajectory points in the past/future in a 2 second window, spanning from -1s to 1s relative to the root coordinate of the frame $i-1$. $\mathbf{t}^a_{i-1} \in \mathbb{R}^{7t}$ denotes continuous action labels changing from zero to one on each of the t trajectory points. This purely auto-regressive vector can be understood as describing the character state and is composed of seven types of motion, representing idle, walk, run, sit, open, carry and climb.
- **Goal Input** $\mathbf{G}_i = \left\{\mathbf{g}^p_{i-1}, \mathbf{g}^d_{i-1}, \mathbf{g}^a_{i-1}\right\}$, is composed of the following items:
  - *Goal Positions and Orientations:* $\mathbf{g}^p_{i-1} \in \mathbb{R}^{3t}$, $\mathbf{g}^d_{i-1} \in \mathbb{R}^{2t}$ are the set of goal positions in the 3D world and directions in the 2D horizontal plane along t ($t = 13$) sampled trajectory points relative to the root coordinate of frame $i-1$. In a low-level control mode where the user controls the locomotion, each goal sample lives 1 second in the future relative to each trajectory point, and thus spans a 2 second window over the future starting from the current frame, which directs the updates of the root trajectory (the goal points stay on the ground in this case). In a high-level control mode for interaction tasks, the goal locations are given by the origin of the objects to interact with. Each goal point represents the target goal location with respect to frame $i-6, \ldots, i+6$.
  - *Action at the Goal:* $\mathbf{g}^a_{i-1} \in \mathbb{R}^{6t}$ denotes one-hot action labels to be launched at each goal point. Note that the climb action is excluded from the goal action as it is difficult to specify the climb to happen at the right timing.

The Interaction Geometry Input $\mathbf{I}_i \in \mathbb{R}^{512}$ and **Environment Geometry Encoder Input** $\mathbf{E}_i \in \mathbb{R}^{1408}$ are both volumetric representations which surround the interaction object with the dimension of 512 and the character with the dimension of 1408, respectively (see Section 4.3).

Outputs:

The output of the Motion Prediction Network $\mathbf{Y}_i = \left\{\mathbf{j}_i^p, \mathbf{j}_i^r, \mathbf{j}_i^v, \tilde{\mathbf{j}}_i^p, \mathbf{t}_i^p, \mathbf{t}_i^d, \mathbf{t}_i^a, \tilde{\mathbf{t}}_i^p, \tilde{\mathbf{t}}_i^d, \dot{\mathbf{g}}_i^p, \dot{\mathbf{g}}_i^d, \dot{\mathbf{g}}_i^a, \mathbf{c}_i, \dot{\mathbf{p}}_i\right\}$ consists of the following components.

- **Predicted Character Pose in Egocentric Coordinate System:** $\mathbf{j}_i^p \in \mathbb{R}^{3j}$, $\mathbf{j}_i^r \in \mathbb{R}^{6j}$ and $\mathbf{j}_i^v \in \mathbb{R}^{3j}$ are the predicted joint positions, rotations and velocities relative to the root coordinate of the frame i.
- **Predicted Character Joint Position in the Future Egocentric Coordinate System:** $\tilde{\mathbf{j}}_i^p \in \mathbb{R}^{3j}$ is the predicted joint positions in the predicted future egocentric coordinate system 1 second ahead, which is defined using $\mathbf{t}_i^p$ and $\mathbf{t}_i^d$ described below.
- **Future Root Trajectories in Egocentric Coordinate System:** $\mathbf{t}_i^p \in \mathbb{R}^{2t'}$, $\mathbf{t}_i^d \in \mathbb{R}^{2t'}$ are the root coordinate positions, forward directions of the future t' ($t' = 6$) trajectory points relative to the root coordinate, and $\mathbf{t}_i^a \in \mathbb{R}^{7t'}$ is the continuous action labels, of frame i in a 1 second window.
- **Future Root Trajectories in Goal-Centric Coordinate System:** $\tilde{\mathbf{t}}_i^p \in \mathbb{R}^{2t'}$, $\tilde{\mathbf{t}}_i^d \in \mathbb{R}^{2t'}$ are the same features as above but are defined relative to the goal-centric coordinate system.
- **Goal Output** $\mathbf{G}_o = \left\{\dot{\mathbf{g}}_i^p, \dot{\mathbf{g}}_i^d, \dot{\mathbf{g}}_i^a\right\}$, $\dot{\mathbf{g}}_i^p \in \mathbb{R}^{3t}$, $\dot{\mathbf{g}}_i^d \in \mathbb{R}^{2t}$ $\dot{\mathbf{g}}_i^a \in \mathbb{R}^{6t}$ are the updated goal parameters.
- **Contact Labels:** $\mathbf{c}_i \in \mathbb{R}^5$ is the key joint contact labels where the key joints are feet, hands and hip.
- **Update in Phase:** $\dot{\mathbf{p}}_i \in \mathbb{R}$ is the 1-dimensional angular phase update.

Auto-regression. The outputs are used to animate the character, directly fed back into the inputs or blended with the user inputs to be a control signal for the next iteration. The character pose is fed back into the network as the frame input in the next iteration. The future root trajectories in egocentric/goal coordinate system are used to compute the future root trajectory as described in Section 4.2. The contact labels can be used when applying IK as post-processing. The phase is updated and passed to the Gating Network in the next round as described in the next section. The Goal Output is blended with the user control signal to be the Goal Input in the next iteration (see Section 4.1).

Network Operation: The operation of Neural State Machine, denoted here by $\mathbf{\Theta}(\cdot)$, is defined as:

$$\mathbf{\Theta}(\mathbf{X}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{W}_2 \text{ ELU}(\ \mathbf{W}_1 \text{ ELU}(\ \mathbf{W}_0\ \mathbf{H}(\mathbf{X}; \boldsymbol{\beta}) + \mathbf{b}_0) + \mathbf{b}_1) + \mathbf{b}_2 \quad (1)$$

where $\mathbf{H}(\mathbf{X}; \boldsymbol{\beta})$ denotes the encoding operation with the Frame Encoder $\mathcal{F}(\cdot)$, Goal Encoder $\mathcal{G}(\cdot)$, Interaction Geometry Encoder $\mathcal{I}(\cdot)$ and Environment Geometry Encoder $\mathcal{E}(\cdot)$:

$$\mathbf{H}(\mathbf{X}; \boldsymbol{\beta}) = \left\{\mathcal{F}(\mathbf{F}_i; \boldsymbol{\beta}^{\mathcal{F}}), \mathcal{G}(\mathbf{G}_i; \boldsymbol{\beta}^{\mathcal{G}}), \mathcal{I}(\mathbf{I}_i; \boldsymbol{\beta}^{\mathcal{I}}), \mathcal{E}(\mathbf{E}_i; \boldsymbol{\beta}^{\mathcal{E}})\right\} \quad (2)$$

where each of these encoders is a simple three-layer fully-connected network, with the network parameters denoted as $\boldsymbol{\beta}^{\mathcal{F}}, \boldsymbol{\beta}^{\mathcal{G}}, \boldsymbol{\beta}^{\mathcal{I}}, \boldsymbol{\beta}^{\mathcal{E}}$ and the dimensions of the hidden/output units set to 512, 128, 256 and 512, respectively.

The parameters of the Motion Prediction Network $\boldsymbol{\alpha}$ are defined by $\boldsymbol{\alpha} = \{\mathbf{W_0} \in \mathbb{R}^{h \times n}, \mathbf{W_1} \in \mathbb{R}^{h \times h}, \mathbf{W_2} \in \mathbb{R}^{m \times h}, \mathbf{b_0} \in \mathbb{R}^h, \mathbf{b_1} \in \mathbb{R}^h, \mathbf{b_2} \in \mathbb{R}^m\}$, where the dimension of the hidden layer h is set as 512, n is the input dimension and m is the output dimension. Furthermore, the weights $\boldsymbol{\alpha}$ are computed by blending K expert weights $\boldsymbol{\gamma} = \{\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K\}$, each of which is in a form of neural network weight configuration:

$$\boldsymbol{\alpha} = \sum_{i=1}^{K} \omega_i \boldsymbol{\alpha}_i, \quad (3)$$

where K is a hyperparameter that can be adjusted according to the complexity and size of the training data, and $\omega = \{\omega_1, ..., \omega_K\}$ are the blending coefficients computed by gating network described next. As we have a wide variation of motion, we set $K = 8$ or $K = 10$ in our experiments, which can cover a wide range of motion types.

The activation function used in Neural State Machine is the exponential rectified linear function [Clevert et al. 2015] defined by

$$\text{ELU}(x) = \max(x, 0) + \exp(\min(x, 0)) - 1. \quad (4)$$

3.2 Gating Network

The Gating Network (see Fig. 2, left) is a three layer fully connected neural network which outputs ω representing the blending coefficients of the experts.

Input Features of the Gating Network: We design the input features of the gating network such that the expert weights are selected and interpolated according to both the action labels and phase values. At each frame i, the input of the Gating Network $\hat{\mathbf{X}}_i$ is computed by composing two vectors as follows:

$$\hat{\mathbf{X}}_i = \mathbf{P}_i \otimes \mathbf{X}'_i, \quad (5)$$

where $\otimes$ is the Kronecker product operation. The first vector is the 2D phase vector $\mathbf{P}_i$:

$$\mathbf{P}_i = \{\sin(\mathbf{p}_i), \cos(\mathbf{p}_i)\} \in \mathbb{R}^2, \quad (6)$$

where the $\mathbf{p}_i$ is a labeled phase scalar in the range of $0 \leqslant \mathbf{p}_i \leqslant 2\pi$. The second vector $\mathbf{X}'_i$ is derived from a subset of the inputs from the Motion Prediction Network including the current action $\mathbf{t}_{i-1}^a$, goal position $\mathbf{g}_{i-1}^p$, goal orientation $\mathbf{g}_{i-1}^d$ and goal action $\mathbf{g}_{i-1}^a$ along the whole trajectory. Further, we found that calculating the distance scalar δ to the goal position $\mathbf{g}_{i-1}^p$ and the angular scalar θ to the goal orientation $\mathbf{g}_{i-1}^d$ and multiplying them with the goal action vector respectively gives the best performance when synthesizing motions for multiple tasks, especially for the foot step planning during approaching of the goals:

$$\mathbf{X}'_i = \left\{\mathbf{t}_{i-1}^a, \mathbf{g}_{i-1}^a, \delta \cdot \mathbf{g}_{i-1}^a, \theta \cdot \mathbf{g}_{i-1}^a\right\} \in \mathbb{R}^{n'}. \quad (7)$$

The key motivation for this input vector design is to let the neural network learn distinct cyclic/non-cyclic phase functions suitable for different actions and goals, instead of using a single, fixed cyclic function as in PFNN [Holden et al. 2017]. This results in an action-specific dynamic blending of expert weights, allowing a single model to encode vastly different motions and still be able to generate sharp motions.

Note that representing the phase as a 2D vector is required since our model directly learns from the phase, which requires a continuous differentiable space that would not be given by feeding a 1D phase scalar progressing from zero to one. This is different from

the original setup used in the PFNN [Holden et al. 2017], where the phase is an external control variable that is first given to a predefined cyclic function before interpolating the network weights. The usage of Kronecker product increases the influence of the phase earlier in the network.

Network Operation: The operation of the Gating Network can then be written as:

$$\omega = \Omega(\hat{\mathbf{X}}; \mu) = \sigma(\mathbf{W}'_2 \text{ ELU}(\mathbf{W}'_1 \text{ ELU}(\mathbf{W}'_0 \hat{\mathbf{X}} + \mathbf{b}'_0) + \mathbf{b}'_1) + \mathbf{b}'_2), \quad (8)$$

where ω is the blending coefficients of the expert weights (see Eq. (3)) to compute the network parameters of the Motion Prediction Network; $\mu = \{\mathbf{W}'_0 \in \mathbb{R}^{h' \times n'}, \mathbf{W}'_1 \in \mathbb{R}^{h' \times h'}, \mathbf{W}'_2 \in \mathbb{R}^{K \times h'}, \mathbf{b}'_0 \in \mathbb{R}^{h'}, \mathbf{b}'_1 \in \mathbb{R}^{h'}, \mathbf{b}'_2 \in \mathbb{R}^{K}\}$ where h' is the number of hidden layer units which is set to 128, n' is the input dimension and $\sigma(\cdot)$ is a softmax operator that normalizes the inputs such that they sum up to 1.

3.3 Network Training

The input and the output of the full dataset are organized in the format of $\mathbf{X} = [\mathbf{X}_1\mathbf{X}_2...], \hat{\mathbf{X}} = [\hat{\mathbf{X}}_1\hat{\mathbf{X}}_2...]$ and $\mathbf{Y} = [\mathbf{Y}_1\mathbf{Y}_2...]$ which are later normalized by the mean and the standard deviation. The network is trained in an end-to-end fashion with the mean squared error cost function:

$$\text{Cost}(\mathbf{X}, \mathbf{Y}; \boldsymbol{\gamma}, \boldsymbol{\beta}, \mu) = \|\mathbf{Y} - \boldsymbol{\Theta}(\mathbf{H}(\mathbf{X}; \boldsymbol{\beta}), \Omega(\hat{\mathbf{X}}; \mu); \boldsymbol{\gamma})\|_2^2. \quad (9)$$

The model is implemented in Tensorflow. We use AdamWR optimizer with the warm restart technique. The learning rate is initialized with the value of $1.0 \cdot 10^{-4}$ which will later decrease controlled by the weight decay rate with the initial value of $2.5 \cdot 10^{-3}$.

4 GOAL-DRIVEN MOTION SYNTHESIS

With the Neural State Machine as the core machinery, we achieve the smooth transitions in high-level states with a goal-driven control while realizing precise scene interactions with a bi-directional character control scheme and volumetric sensors.

4.1 Goal-Driven Character Control

Our system has two modes of control: the high-level goal-driven mode and the low-level locomotion mode. In the goal-driven mode, the user selects the target object (chair, door or object) by the mouse and can select the desired action to perform with the object via keyboard buttons. The character automatically moves toward the object and launches the required series of actions (sit, open or carry) at the right timing. In the locomotion mode, the user interactively controls the character by the keyboard or the gamepad, and the character walks/runs in the direction instructed by the user.

Our system can seamlessly switch between the low-level locomotion mode and the high-level goal-driven mode through Goal Inputs $\mathbf{G}_i$: in the locomotion mode, the Goal Outputs $\mathbf{G}_O$ are blended with the trajectory produced by the user control signal in a 1s time window and fed back into the network as the Goal Inputs $\mathbf{G}_i$, as done in [Holden et al. 2017]. In the goal-driven mode, the $\mathbf{G}_i$ is set to the object/environment of the user's selection. $\mathbf{G}_i$ can be changed at any time and valid transitions can be naturally produced based on pose of the character and the foot stepping pattern.

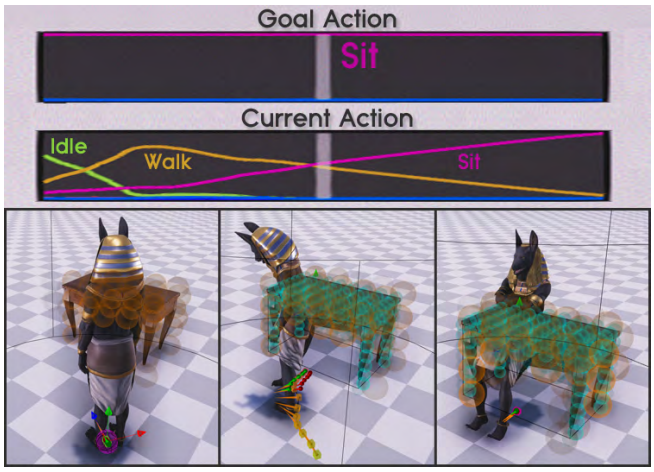

Fig. 3. Neural State Machine with goal-driven control enables smooth transitions between high-level states. Given a single goal action, the network automatically predicts the necessary intermediate states to reach the goal state smoothly.

We show a usage of the high-level controller for a sitting down task in Fig. 3. The character needs to start from an idle state, transition from idle to walk until getting close to the chair, and then finally transition from walk to sit. Synthesizing such series of movements only by the low-level locomotion control requires meticulous user input. In contrast, when using the high-level controller, the user only needs to indicate "sitting on the chair" as the desired goal action (sitting in case of Fig. 3). This is then fed into the Gating Network, which activates the required expert weights for producing the motion in the next frame. The Motion Prediction Network then produces the action label every frame; in the case of Fig. 3, first the transition from idle to the walk, and then increasingly the action value for sitting down once the character is close enough to the object.

4.2 Bi-Directional Control Scheme

The idea of the bi-directional controller is to infer the motion from both the egocentric and goal point of view, match both predictions during runtime, and feedback such inference into the Neural State Machine to increase the precision of the character to reach the goal during the tasks (Fig. 4). When using only an egocentric controller that is often used for real-time character controllers, there is a difficulty in reaching the goal location and conducting the task. Intuitively, as the egocentric perspective relies on a root update to bring the character closer to the target, there is an error accumulation over multiple frames that alone can not guarantee termination at the goal location. Instead, predicting such information in the goal-space gives more accurate values for where the character would actually need to be, and back-transforming those into root-space during runtime enables avoidance of such error accumulation. Especially when the variation of the training data is not very large, we find the system is more likely to not reach the goal with high precision from random locations when not using such a bi-directional inference scheme.

Fig. 4. A bi-directional trajectory prediction improves the accuracy in reaching the desired goal location. The future trajectory is predicted both relative to the character root (red trajectory) as well as to the coordinate frame of the goal (green curve), and interpolated before being given as the input for the next frame.

In more detail, the bi-directional controller is part of the Neural State Machine and computes the future trajectory in the goal-centric coordinate system (see *Outputs* of Section 3.1):

$$\left\{\mathbf{t}_i^p, \mathbf{t}_i^d, \tilde{\mathbf{t}}_i^p, \tilde{\mathbf{t}}_i^d\right\} = \hat{\mathbf{\Theta}}(\mathbf{X}_i),$$

where $\{\mathbf{t}_i^p, \mathbf{t}_i^d\}$ and $\{\tilde{\mathbf{t}}_i^p, \tilde{\mathbf{t}}_i^d\}$ are the future root trajectories in egocentric/goal coordinate system, and $\hat{\mathbf{\Theta}}(\cdot)$ is a sub-function that denotes the bi-directional controller part of the Neural State Machine. $\{\tilde{\mathbf{t}}_i^p, \tilde{\mathbf{t}}_i^d\}$ are then converted back to the egocentric coordinate system by multiplying the goal-to-egocentric transformation matrix and blending it with the corresponding egocentric outputs according to a blending parameter $\lambda : (1-\lambda)\{\mathbf{t}_i^p, \mathbf{t}_i^d\} + \lambda \mathbf{R}_i\{\tilde{\mathbf{t}}_i^p, \tilde{\mathbf{t}}_i^d\}$, where $\mathbf{R}_i$ is the goal-to-egocentric transformation matrix. The blending parameter $\lambda = w_i{}^{d_i^2}$ is computed in such a way that it has larger effect when the target is closer and vanishing effect when further away, where w_i is a linearly increasing weight between 0 and 1 from the current to last trajectory sample, and d_i^2 is the squared distance of each trajectory sample i to its goal. The computed trajectory is fed back into the Neural State Machine in the next iteration as the future trajectory part of $\{\mathbf{t}_i^p, \mathbf{t}_i^d\}$.

We also adjust the character joint velocities in a bi-directional manner so that it helps reaching the goal – especially for satisfying the contacts with the objects. Here, the velocity in the next frame is revised as $(1-\lambda)\mathbf{j}_i^v + \lambda(\mathbf{R}_i'\tilde{\mathbf{j}}_i^p - \mathbf{j}_i^p)$, where the blending parameter λ is chosen by the delta time Δt between two consecutive frames, and $\mathbf{R}_i'$ is the transformation from the predicted egocentric coordinate system at frame $i{+}t'$ to that of i. Intuitively, this adjusts the velocities to the direction of where the character posture would need to be in the future, and smoothly drives the character towards the accurate pose as presented in Section 7.

4.3 Volumetric Sensors

We use two voluemtric sensors to evaluate the status of the body with respect to the object: the Environment Sensor (see Fig. 5, bottom) and the Interaction Sensor (see Fig. 5, right-top)

Environment Sensor. To recognize the surrounding geometry of the character and let it affect the motion in the next frame, we use a volumetric sensor that we call the Environment Sensor that has a cylindrical shape. The collision between objects/environment and a cylindrical volume of radius R and height H is evaluated while the character is moving (the orange spheres in Fig. 5) and fed into the Environment Geometry Input $\mathbf{E}_i$. Within the volume, spheres of radius $r << R$ are sampled and their intersections with the objects/environment are tested. The spheres are sampled in a polar fashion at the resolution of $(\lceil \frac{R}{2r} \rceil, \lceil \frac{\pi R}{r} \rceil, \lceil \frac{H}{r} \rceil)$, along the radius, arc and height direction.

Within each sphere cell, we compute a floating value that roughly corresponds to the occupancy by the external objects/environment; for this purpose, we prepare a coarse collider object composed of cuboids (see Fig. 5, top-middle). The sensor value is computed by:

$$s = \begin{cases} 1 & \text{if sphere center inside the collider} \\ 0 & \text{if sphere not intersecting collider} \\ 1 - \frac{d}{r} & \text{otherwise} \end{cases}$$

where d is the closest distance between the collider and the sphere center, and r is the radius of the sphere. This setup provides continuous inputs into the volumetric sensor and thus results in smooth movements. The collider object is computed automatically from a high resolution voxel model that is generated from the object mesh. The adjacent voxels are merged to form cuboids, and the cuboids

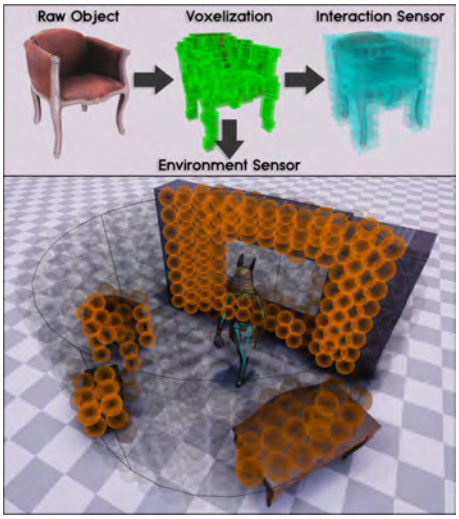

Fig. 5. Geometry Processing Pipeline: Given the mesh, we first approximate its collision geometry via voxelization, which is then used by the environment and interaction sensors to extract the surrounding scene information.

are iteratively merged until the merger does not form a cuboid any more. The sensor values are fed into the Environment Geometry Encoder Input $\mathbf{E}_i$. The usage of the Environment Sensor induces the character to launch motion to interact with/avoid the object when it appears in the vicinity.

Interaction Sensor. We prepare another volumetric sensor that we call the Interaction Sensor to provide further details of the object geometry from the goal point of view (see Fig. 5, top-right). As the Environment Sensor can be too coarse, and the character can be moving, the fine details of the object, such as the arm rests of the chair, can be missed. The Interaction Sensor is an $8 \times 8 \times 8$ cubic volume that covers the target object. Each element contains a 4D vector. The first three elements are the relative vector of the body root from the element center. The last element s' is computed by

$$s' = \begin{cases} 1 & \text{if voxel center inside the collider} \\ 0 & \text{if voxel not intersecting collider} \\ 1 - \frac{d'}{r'} & \text{otherwise} \end{cases}$$

where d' is the shortest distance between the center and the collider, and r' is the diagonal length of the voxel. These inputs are fed into the Interaction Geometry Input $\mathbf{I}_i$. The usage of the Interaction Sensor improves the precision of the interaction as we show in our evaluation (see Section 7). For animating the character to carry or place the box, we dynamically adjust the position values of the Interaction Sensor according to the Goal Input $\mathbf{G}_i$; their values are computed based on the future goal position and orientation at $t = 6$, which is 1 second ahead of the current time frame.

5 DATA PREPARATION

In this section, we describe the data preparation, especially the motion capture, the data labelling, the 3D object/environment fitting and its data augmentation procedure.

5.1 Motion Capture and Labelling

We build prop environments to capture a dataset of a person approaching and sitting on a chair, avoiding obstacles while walking, opening and passing through a door, picking up, carrying and putting down a box. We also build some more complex situations with a concave environment like sitting on a chair that is partially under the desk, so that the character needs to stretch their legs or put their hands on the desk to complete the sitting motion. A total of 94 minutes of motion capture data is captured by an XSens inertia motion capture system and exported. The size of the data is later doubled by mirroring.

Label	Seconds	%
Idle	1603	26.7
Walk	2600	43.3
Run	354	5.9
Sit	1333	22.2
Open	126	2.1
Carry	462	7.7
Climb	114	1.9

Table 1. Action distribution of motion capture.

Each frame of data is then annotated with one of the action labels, "idle", "walk", "run", "sit", "open", "climb" or "carry", or a combination of two ("carry+idle", "carry+walk"), which describes the motion type at the current frame. The distribution of the action labels for the un-mirrored dataset is summarized in Table 1. Apart from the action label at the current frame, we also label a goal action for each frame, which is actually the action labels in the future frames. By training the system using the goal labels as the part of Goal Inputs $\mathbf{G}_i$, the user can interactively control the character by specifying the goal action during runtime.

Finally, we label a phase scalar for each of the frames in the dataset. For locomotion, the phase is defined as 0, π and 2π according to when the left/right/left foot lands on the ground, and the phases in between are computed by interpolation. For the acyclic motions like sitting, the phase is defined as 0 and 2π for the frame where the transition starts and ends, and the middle part is again computed by interpolation.

5.2 Object/Environment Fitting and Data Augmentation

In this section, we describe the object/environment fitting process and how we do data augmention, so that the system can adapt the character motion to a wide range of objects and environments without increasing the training data size. For achieving this goal, the space of motion with respect to the shapes needs to be well covered. Naively, this would require capturing all the subtle and larger variations of motion and shape modifications in such environments, leading to an enormous effort not only for motion capture and setting up objects in the real world, but also for data labeling, fitting and training. Having this in mind, we design a motion augmentation framework that allows us to enrich the dataset with minimum data size explosion by editing both the motion and geometry while preserving the context. After fitting a template object into the motion capture data, at each frame, we randomly sample new objects from the pool of objects/environments obtained from ShapeNet [Chang et al. 2015] and adapt the pose following the idea of relationship descriptors [Al-Asqhar et al. 2013] such that the context is preserved.

The scheme is composed of the following five steps (also see Fig. 6):

Step 1: Object/Environment Fitting: We fit a template object into one of the motion captures using a frame where the body is static with respect to the object. This only requires specifying the location and orientation of the object at one frame, where the body is static with respect to the object.

Step 2: Locating the Contact Points and Redefining the Key Joint Trajectories: Contact points are embedded on the object / environment surface when the key joints are within a threshold distance of the surface. Also, the position of the key joints are redefined as the end points of the relative vectors originating from the contact points.

Step 3: Switching/Transforming the Object/Environment: When the objects are switched or transformed, either through translation,

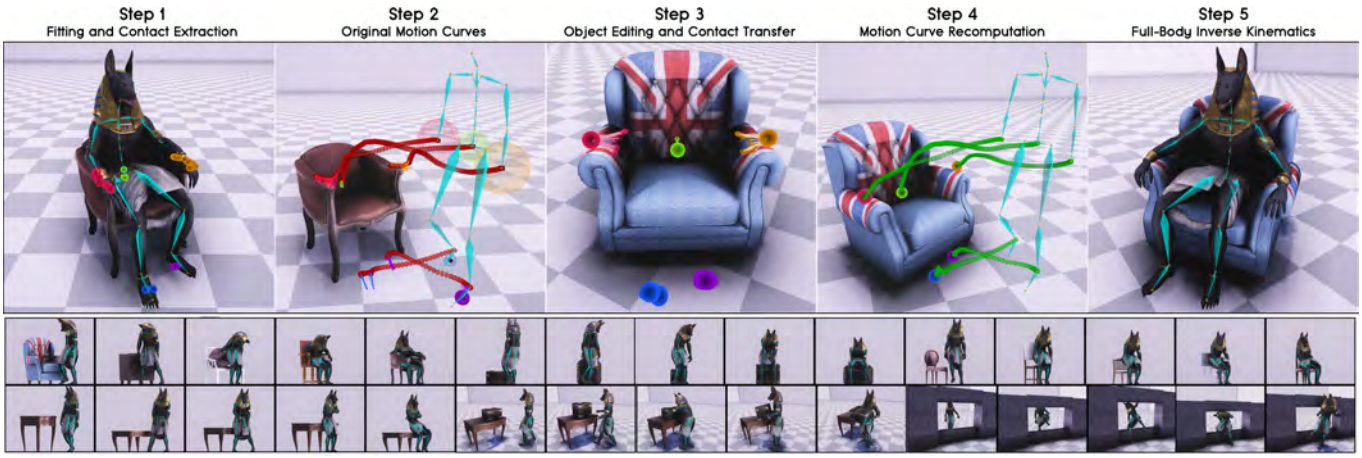

Fig. 6. The five steps of data augmentation for adapting to different shapes (top). Step 3-5 are applied per frame to prepare training data where the character can adapt to different shapes without increasing the data size (bottom).

rotation or scaling, the position of the contact points are reset. When the object is transformed, the contact points simply follow the transformation that was applied. When the object is switched, we initially project the template contact points to the nearest surface. When this fails, we manually specify the new location on the object surface. This semi-automatic process is possible as we use a rather small set of training data (20 chairs, 2 desks, 3 boxes, 2 windows). Note that we only use a single position for the new contact point per key joint while there could be multiple in the template object.

Step 4: Updating the Key Joint Trajectories: The key joint positions are recomputed using the updated contact points and the relative vectors for frames where the key joints were within the threshold in Step 2. Otherwise, they are blended between the original motion and the adapted positions with a 1 second time window.

Step 5: Full-Body Pose Computation: Finally, the full character pose is recomputed by full-body IK based on the CCD framework. Starting from the motion capture pose, the root translation and orientation is first computed such that the error of the end effectors is minimized, and then going down the hierarchy to minimize the error of each joint's end effectors. Although this does not guarantee temporal continuity, we find it stable and it produces smooth movements. The computed pose together with the new object is used as training data for the neural network.

When preparing the training set for the system, for every frame of the training data we randomly switch/transform the object or environment by applying step 3-5 (see Fig. 6, bottom). Such a random sampling allows us to prepare enough variation to cover the wide shape/pose space. We find this approach much more efficient compared to preparing the entire motion set for all types of objects and using them as the training data, where the training time increases in the order of (motion number × object number) when doing full training with all objects.

6 EXPERIMENTS

In this section, we describe the results of animating different character movements with interaction with objects and the environment in daily life, using the Neural State Machine.

Data Pre-processing and Training. The data is preprocessed to be in the form of input $\mathbf{X}$ and output $\mathbf{Y}$ for the Neural State Machine. The size of the training data is 16GB. The full training process described in Section 5 takes 70 epochs where the learning rate is reset at epoch 11 and 31. The batch size is set to 32 and during training the batches are randomly sampled. The full training of 70 epochs takes around 20 hours on an NVIDIA GeForce GTX 1080Ti GPU. After the training, the network size is around 250MB.

Animated Results. Figure 8 shows results of multiple tasks achieved by one trained model of Neural State Machine. We have successfully trained it to sit, avoid obstacles, open doors, and carry objects along with standard locomotion. The readers are referred to the supplementary video for the details.

The sitting task has been trained with various types of chairs. The system is able to generate the appropriate motion based on the chair

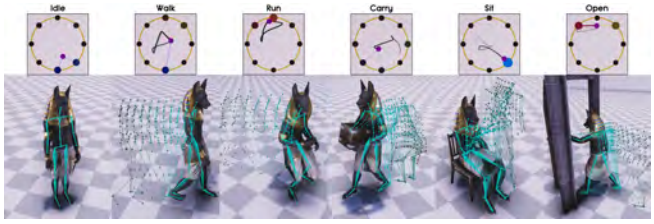

Fig. 7. Visualization of the learned phase cycle for each task. Black dots on the yellow circle represent expert neural networks which are blended to generate the motion prediction network. The trajectory inside the circle is determined by the activation of the experts. The distinct trajectory for each task indicates how well they are separated in the motion space to avoid undesired blending.

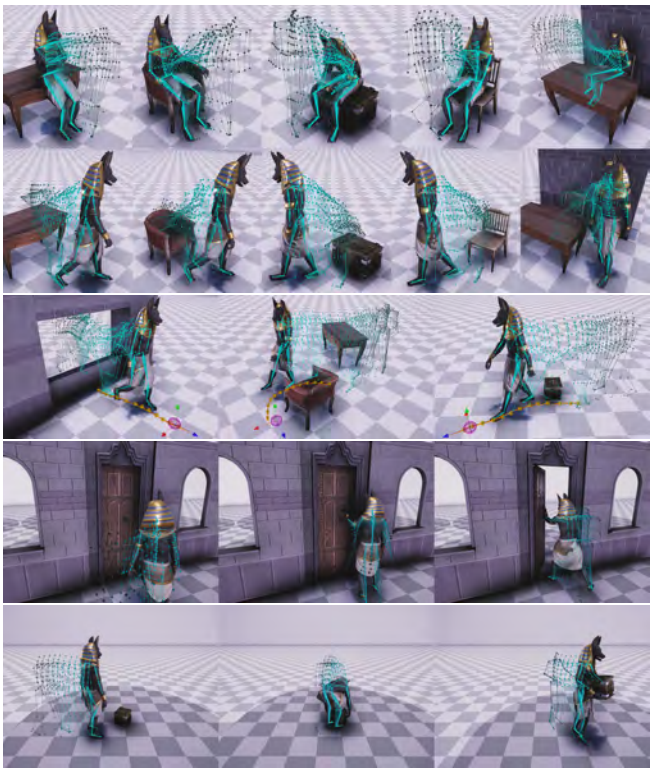

Fig. 8. Results of Neural State Machine generating motion for various tasks. First and second rows: various sitting tasks adapting to size and to obstacles like a desk. Third row: collision avoidance tasks including concave objects. Fourth row: opening door. Fifth row: carrying objects.

geometry. The system has also learned to properly slide in sideways to sit on a chair to avoid a desk in front of it. This demonstrates how the volumetric geometry input is able to encode the concavity of the colliding objects. The avoidance task has been successfully trained to go through a concave geometry like a wall with a hole. Again, this shows the strength of the volumetric geometry input encoding the concave geometry. This would not have been possible with a naive scene encoding with height maps. The open door and carry object tasks demonstrate successful motion planning to precisely reach the scene objects at the right orientation. This is due to the effectiveness of the goal-guided controller.

Figure 7 visualizes how different tasks are encoded by our learning framework. The system cycles through different experts for each action, and the distinct shapes of the phase trajectory for each task indicate how well different tasks become separated in the model.

Testing with Novel Objects. We show the results of applying our approach for sitting on novel chairs in Fig. 9. The data augmentation allows our system to let the character sit on chairs that have not be seen before. Despite the wide variety of geometry and sizes, the character can sit on them and bring the arms to the arm rest if the chair has them.

Technique	Idle to Walk	Idle to Run	Walk to Sit	Run to Sit
MLP	1.75	1.35	5.05	5.13
PFNN	1.50	0.87	4.95	5.32
MANN	1.83	0.92	**4.85**	5.30
LSTM	2.16	2.24	5.62	-
Auto-LSTM	1.86	1.82	5.53	-
NSM (Ours)	**1.49**	**0.81**	4.92	**5.10**

Table 2. The average response time in seconds produced by different models. The responsiveness from idle to walking/running is measured by the style transition time to 4 different directions (forward/backward/left/right). The responsiveness from walking/running to sitting is measured by the task accomplishing time from 10 different positions/directions within 3 meters from the chair.

7 EVALUATION

In this section, we first compare the results computed by the Neural State Machine with other alternative time series models. We next present the results of an ablation study that justifies the design/components of our system. The readers are referred to the supplementary video for the visual comparison.

7.1 Comparison with Other Time-Series Models

We compare our method against several alternative methods in terms of responsiveness, precision and foot-sliding artifacts.

The responsiveness (see Table 2) is measured by the time required to transition to the target action label. The precision (see Table 3) is measured by the rotational error (RE) and positional error (PE) when reaching the goal location, when the characters start from the same starting position. The foot-sliding (see Figure 10) is measured by accumulating the drift of feet in the horizontal plane when the distance betwen the feet and the floor is lower than a certain threshold (2.5 *cm*). The other architectures that we compared our approach

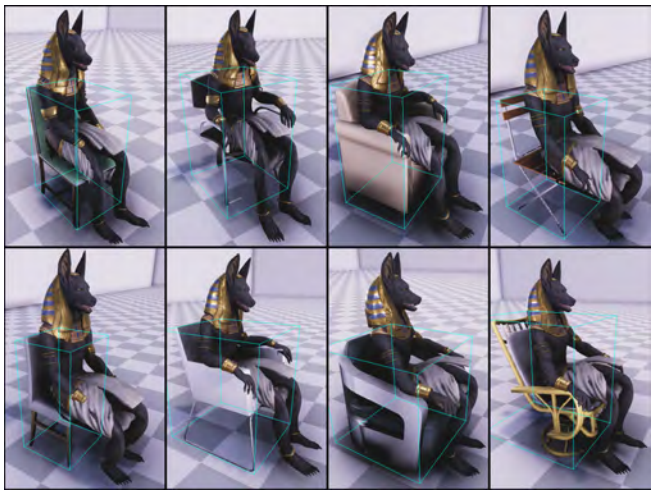

Fig. 9. Results of successfully sitting on novel chairs that are not used for training.

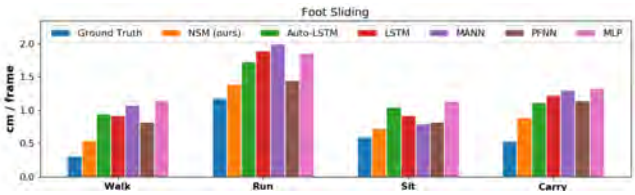

Fig. 10. The average foot sliding produced by different models. Sliding in the walking and running tasks is measured in different moving directions. Sliding in the sitting task is measured during approaching and sitting transitions. Sliding in the carrying task is measured during approaching and holding movements.

against include feed-forward neural network (MLP), PFNN, MANN and LSTM. We set the complexity of the models similar to ours; use the same set of input/output data for a fair comparison.

Overall, our system performs the best, although other models perform equally well for some criteria. We now compare our approach with the individual alternatives. Note that when we evaluate the Neural State Machine using the sitting action, we use a test chair that is not included in the training data.

Comparison with LSTM. When we train an LSTM model that has a similar structure with [Lee et al. 2018] (4 recurrent layers, encoder and decoder structure), the system performs well except that there are some issues with responding to user inputs. The character mostly only starts to move towards the target object after walking for a few steps. Also, the character cannot directly transition to states that are not observed in the training data: e.g., transition from running to sitting completely fails. For making a controller responsive to user inputs, the system needs to learn the character should start approaching the goal irrespective of the memory state. In fact, Lee et al. [2018] produce a large amount of augmented data by editing a short locomotion clip by Laplacian editing. In our scenario, where the character needs to approach and precisely reach a goal, humans use different stepping pattern according to the way they approach,

	Sit		Carry	
Technique	PE(cm)	RE(deg)	PE(cm)	RE(deg)
MLP	14.05	4.11	6.27	3.46
PFNN	7.66	3.31	10.35	4.83
MANN	5.54	1.87	3.41	1.79
LSTM	6.14	2.05	6.11	3.02
Auto-LSTM	5.23	2.70	5.59	3.77
NSM (Ours)	**2.93**	**0.92**	**3.05**	**1.02**
Bi-Directional	4.27	1.58	3.41	1.73
Interaction Sensor	3.72	1.11	6.09	2.26
Environment Sensor	3.05	0.94	8.52	2.27

Table 3. Upper and middle parts: The average positional error (PE) and rotational error (RE) produced by different models in the sitting and carrying tasks. Lower part: The error produced by NSM when the corresponding technique is removed. 10 different character positions/rotations are initialized for each task, error is measured at the ending point of the sitting and grasping transitions on the hip and hand joints correspondingly.

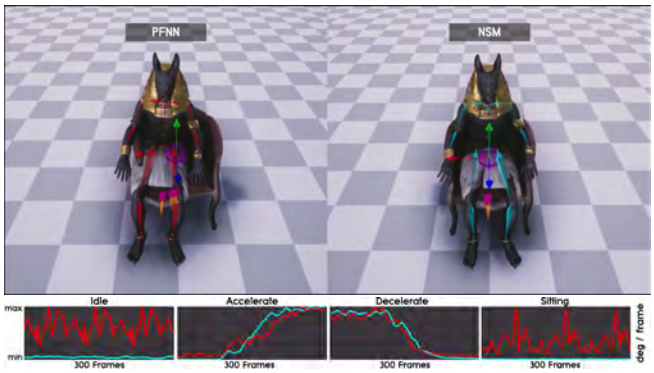

Fig. 11. Left: a character driven by the PFNN penetrating a chair and exhibiting vibrating artifacts because it blends motions in drastically different tasks. Right: Neural State Machine architecture precisely aligns the character to the chair without vibrating artifacts because it only blends the motions from the relevant task. In the graphs below, we show the normalized average joint angle updates for each motion, where can be observed that such vibrating artifacts on the PFNN exist during all motion (also see supplementary video).

so editing a single example is not a wise choice. Similarly, LSTM fails to conduct a transition that is not observed in the training, such as running to sitting.

We also adopt the training strategy of probabilistic teacher forcing (Auto-LSTM) [Li et al. 2017] which uses model output from a prior time step as an input with a certain probability. Although such a training technique improves the responsiveness, the character still fails to perform some transitions where less training data exists.

These results show that our approach performs better than LSTM especially for responsiveness when the locomotion data set is not heavily augmented.

Comparison with PFNN. PFNN has much more restrictions compared to NSM for synthesizing character-scene interactions. With highly multi-modal data from different interaction behaviors, the PFNN results in unwanted blending of poses leading to imprecise output with vibrating artifacts, as shown in Figure 11. Note that the PFNN uses a single cyclic phase function where the control points are interpolated by the phase. The phase values are defined based on contacts. This works well for locomotion where the contacts follow a consistent foot-fall pattern. However the contacts and the phase labels are different between motions such as sitting, carrying and locomotion. For example, for sitting, there is a stationary contact between the back and the chair while locomotion has regular footfall patterns. Possibly due to such inconsistency, the precision (Table 3) for sitting and carrying of the PFNN is much worse than our approach, and also resulting in vibration artifacts. This is in contrast to NSM, where the phase function is learned from the action labels and the phase label. The system learns to use different combinations of control points during the phase cycle, as shown in Fig. 7.

Because the PFNN is designed for locomotion and locomotion takes up a large portion of our data, the responsiveness (Table 2) and foot sliding artifacts (Figure 10) are similar to our approach.

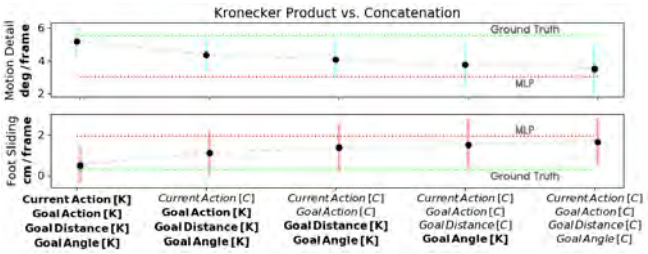

Fig. 12. The resulting performance in motion detail and foot sliding when combining features with the phase by either using Kronecker product [K] or concatenation [C] as input to the gating network.

Comparison with MANN. The Gating Network of the MANN automatically computes the weighting of the expert weights and thus does not require providing the phase variable. The responsiveness (Table 2) of the MANN is equally good or sometimes even better than our approach. Also, the precision (Table 3) of the MANN is much better than the PFNN although not as good as ours. On the contrary, the MANN produces more foot sliding artifacts (Figure 10) compared to the PFNN and our approach, especially when randomly changing directions and switching between different locomotion types. In such moments, the network tends to produce low joint angle updates which causes the whole motion to appear rather stiff. In return, the motion can appears very unresponsive although the root motion responds to the control signals, which leads to the main source of foot sliding artifacts. Those problems could be due to the difficulty of segmenting the motion types only by the foot velocities, which is the feature that the MANN uses for gating.

7.2 Ablation Study

We show an ablation study examining the of precision of sitting to demonstrate the effect of the 2D phase representation with Kronecker product, bi-directional controller, interaction volume input and environment volume input. The results are shown in Table 3, bottom. It can be observed that without this functionality, the precision degrades.

Kronecker Product. In Fig. 12, we measure the motion quality when modulating the gating input features with the phase via the Kronecker product instead of doing basic feature concatenation. The motion detail is measured as the average joint angle update, and foot sliding is measured by the ratio of horizontal foot movement when close above ground, similar to Figure 10. Using the Kronecker product achieves the best performance while including more features via concatenation incrementally decreases quality. Note that when concatenating features, the phase is still used as an input feature in the hope that the network could learn from this representation, but instead this feature seems to increasingly lose importance. This loss in quality can be explained by the weak segmentation characteristic of concatenated features, which often remain constant throughout the motion – particularly for one-hot action labels. When modulated by the phase, the segmentation of motion through time remains well-preserved with respect to the modulated features.

Fig. 13. Top: ablation study for bi-directional control. The left character without trajectory and pose interpolation is misaligned on the chair. Bottom: ablation study for Interaction Sensors. The left character has more difficulty learning from the geometry and is penetrating into the chair with an unnatural pose.

Bi-directional Controller. As can be observed in the top image of Fig. 13, without the bi-directional controller, the character cannot correctly align to the target object. The left character without bi-directional control is off-centered and hands are not placed on the armrests, which results in unnatural poses and object penetration.

Interaction Sensor. Without the Interaction Sensor, the character cannot properly adapt to the target geometry as shown in the bottom image of Fig. 13. The left character without the Interaction Sensor penetrates through the chair in an obscure pose.

Environment Sensor. As the resolution of the Environment Sensor is low, the influence in the error in Table 3 is limited. However, without the Environment Sensor, the character does not interact well with the objects. Also, as can be observed in Fig. 14, the character ignores obstacles in the front of the character when approaching the chair, and penetrates through it to reach to the goal.

8 LIMITATIONS AND FUTURE WORK

As we train the system with only a small data set of 3D geometry, although it works reasonably well for chairs similar to those in the data set, the system fails to adapt to geometry that is rather different from that in the training set (see Fig. 15). To adapt to a wider range of 3D shapes, in addition to increasing the training size, we would need to use a more complex representation, such as higher resolution volume data, or possibly other geometry encoders such as PointNet++ [Qi et al. 2017] pretrained with a large geometry dataset such as ShapeNet. Such a representation can lead to a

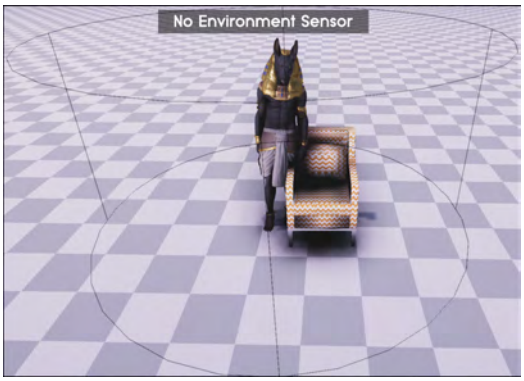

Fig. 14. When the environment sensor is disabled, the character can easily pass through the chair geometry while approaching the object to sit down. Commonly, the produced trajectories can be observed to not maintain a safety distance around the object, and it is likely to observe unrealistic movements, such as hands or legs penetrating through the object.

generalized shape and scene understanding with more convincing motion adaptation.

When switching the goal action, the input into the network can change discontinuously, resulting in abrupt movements. This is more obvious when suddenly activating objects to interact with. Such artifacts can be mitigated by smoothly updating the inputs over a time window. On the contrary, a longer time window can result in slower response, which may not be desirable for real-time applications. We find a window of 0.25-0.5 seconds can produce smooth effects with less noticable latency.

Currently, the type of interactions by our character are limited to sitting, holding and avoiding obstacles. It will be interesting to look into a direction to capture the motion and the environment information together, for example by mounting a RGBD camera on the body of a subject wearing a mocap suit. Such a capture scheme can enrich the data needed for our system without manual intervention.

We found the phase encoded into the input vector produced better motions because it avoids the state machine from getting stuck. The downside of this is all motions require phase labelling.

Fig. 15. The arms fail to stay on the arm rests and the hip floats in the air for a chair drastically different from the training set.

Automatic phase labelling works well for motions with clear cycles but for general acyclic motions we need manual phase labelling with heuristic rules applied across the dataset. A motion may also consist of multiple cyclic movements with different periods, such as a person swinging a sword while walking. Generalizing the detection of multiple phases in motion is one future direction to ease the set-up and improve results.

9 CONCLUSION

We propose a novel neural network framework to synthesize motions that involve close interactions with the environment. Our system can handle a wide variety of motion types and allows the users to guide the characters by only providing abstract goals. The system can handle both cyclic and acyclic movements and can be applied for real-time applications such as computer games and virtual reality systems.

REFERENCES

Shailen Agrawal and Michiel van de Panne. 2016. Task-based Locomotion. *ACM Trans on Graph* 35, 4 (2016). https://doi.org/10.1145/2897824.2925893

Rami Ali Al-Asqhar, Taku Komura, and Myung Geol Choi. 2013. Relationship descriptors for interactive motion adaptation. In *Proc. SCA*. 45–53. https://doi.org/10.1145/2485895.2485905

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018). http://arxiv.org/abs/1803.01271

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015). http://arxiv.org/abs/1512.03012

Myung Geol Choi, Manmyung Kim, Kyung Lyul Hyun, and Jehee Lee. 2011. Deformable motion: Squeezing into cluttered environments. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library. https://doi.org/10.1111/j.1467-8659.2011.01889.x

Min Gyu Choi, Jehee Lee, and Sung Yong Shin. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans on Graph* 22, 2 (2003). https://doi.org/10.1145/636886.636889

Alexander Clegg, Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. 2018. Learning to dress: synthesizing human dressing motion via deep reinforcement learning. *ACM Trans on Graph* 37, 4 (2018). https://doi.org/10.1145/3272127.3275048

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR* abs/1511.07289 (2015). arXiv:1511.07289 http://arxiv.org/abs/1511.07289

Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent network models for human dynamics. In *Proc. ICCV*. 4346–4354. https://doi.org/10.1109/ICCV.2015.494

Félix G. Harvey and Christopher J. Pal. 2018. Recurrent Transition Networks for Character Locomotion. *CoRR* abs/1810.02363 (2018). arXiv:1810.02363 http://arxiv.org/abs/1810.02363

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286* (2017). https://doi.org/10.1145/1833349.1778865

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Trans on Graph* 36, 4 (2017), 42. https://doi.org/10.1145/3072959.3073663

Daniel Holden, Jun Saito, and Taku Komura. 2016. A deep learning framework for character motion synthesis and editing. *ACM Trans on Graph* 35, 4 (2016). https://doi.org/10.1145/2897824.2925975

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural Computation* 3, 1 (1991), 79–87. https://doi.org/10.1162/neco.1991.3.1.79

Manmyung Kim, Youngseok Hwang, Kyunglyul Hyun, and Jehee Lee. 2012. Tiling motion patches. In *Proc. SCA*. 117–126. http://dl.acm.org/citation.cfm?id=2422356.2422375

Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. *ACM Trans on Graph* 21, 3 (2002). https://doi.org/10.1145/566654.566607

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-objective Control. *ACM Trans on Graph* 37, 6 (2018). https://doi.org/10.1145/3272127.3275071

Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. 2006. Motion patches: building blocks for virtual environments annotated with motion data. *ACM Trans on Graph* 25, 3 (2006). https://doi.org/10.1145/1141911.1141972

Zimo Li, Yi Zhou, Shuangjiu Xiao, Chong He, Zeng Huang, and Hao Li. 2017. Auto-conditioned recurrent networks for extended complex human motion synthesis. *arXiv preprint arXiv:1707.05363* (2017). http://arxiv.org/abs/1707.05363

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based contact-rich motion control. *ACM Trans on Graph* 29, 4 (2010), 128. https://doi.org/10.1145/1833349.1778865

Wan-Yen Lo, Claude Knaus, and Matthias Zwicker. 2012. Learning motion controllers with adaptive depth perception. In *Proc. SCA*. 145–154. http://dl.acm.org/citation.cfm?id=2422356.2422378

Wan-Yen Lo and Matthias Zwicker. 2008. Real-time planning for parameterized human motion. In *Proc. SCA*. 29–38. http://dl.acm.org/citation.cfm?id=1632592.1632598

Jianyuan Min and Jinxiang Chai. 2012. Motion Graphs++: A Compact Generative Model for Semantic Motion Analysis and Synthesis. *ACM Trans on Graph* 31, 6 (2012), 153:1–153:12. https://doi.org/10.1145/2366145.2366172

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans on Graph* 31, 4 (2012). https://doi.org/10.1145/2185520.2185539

Tomohiko Mukai and Shigeru Kuriyama. 2005. Geostatistical motion interpolation. *ACM Trans on Graph* 24, 3 (2005). http://doi.acm.org/10.1145/1073204.1073313

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018a. DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills. *ACM Trans on Graph* 37, 4 (2018). https://doi.org/10.1145/3197517.3201311

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Trans on Graph* 35, 4 (2016). https://doi.org/10.1145/2897824.2925881

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans on Graph* 36, 4 (2017). https://doi.org/10.1145/3072959.3073602

Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. 2018b. SFV: reinforcement learning of physical skills from videos. *ACM Trans on Graph* 37, 6 (2018). https://doi.org/10.1145/3272127.3275014

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proc. NIPS*. 5099–5108. http://papers.nips.cc/paper/7095-pointnet-deep-hierarchical-feature-learning-on-point-sets-in-a-metric-space.pdf

Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. 1998. Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Computer Graphics and Applications* 18, 5 (1998), 32–40. http://dx.doi.org/10.1109/38.708559

Charles F Rose III, Peter-Pike J Sloan, and Michael F Cohen. 2001. Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation. *Computer Graphics Forum* 20, 3 (2001), 239–250. https://doi.org/10.1111/1467-8659.00516

Alla Safonova and Jessica K Hodgins. 2007. Construction and optimal search of interpolated motion graphs. *ACM Trans on Graph* 26, 3 (2007). https://doi.org/10.1145/1276377.1276510

Hubert PH Shum, Taku Komura, Masashi Shiraishi, and Shuntaro Yamazaki. 2008. Interaction patches for multi-character animation. *ACM Trans on Graph* 27, 5 (2008). https://doi.org/10.1145/1457515.1409067

Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. 2018. Neural Kinematic Networks for Unsupervised Motion Retargetting. *Proceedings of CVPR 2018*. https://doi.org/10.1109/CVPR.2018.00901

J.M. Wang, D.J. Fleet, and A. Hertzmann. 2008. Gaussian Process Dynamical Models for Human Motion. *IEEE PAMI* 30, 2 (Feb 2008), 283–298. https://doi.org/10.1109/TPAMI.2007.1167

Jungdam Won, Kyungho Lee, Carol O'Sullivan, Jessica K Hodgins, and Jehee Lee. 2014. Generating and ranking diverse multi-character interactions. *ACM Trans on Graph* 33, 6 (2014). https://doi.org/10.1145/2661229.2661271

Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. 2017. How to train your dragon: example-guided control of flapping flight. *ACM Trans on Graph* 36, 6 (2017). https://doi.org/10.1145/3130800.3130833

Yuting Ye and C Karen Liu. 2012. Synthesis of detailed hand manipulations using contact sampling. *ACM Trans on Graph* 31, 4 (2012). https://doi.org/10.1145/2185520.2185537

Wenhao Yu, Greg Turk, and C Karen Liu. 2018. Learning symmetric and low-energy locomotion. *ACM Trans on Graph* 37, 4 (2018). https://doi.org/10.1145/3197517.3201397

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive neural networks for quadruped motion control. *ACM Trans on Graph* 37, 4 (2018). https://doi.org/10.1145/3197517.3201366

Wenping Zhao, Jianjie Zhang, Jianyuan Min, and Jinxiang Chai. 2013. Robust realtime physics-based motion control for human grasping. *ACM Trans on Graph* 32, 6 (2013). https://doi.org/10.1145/2508363.2508412