Data-driven Autocompletion for Keyframe Animation

Xinyi Zhang
Department of Computer Science
University of British Columbia
xinyi92@cs.ubc.ca

Michiel van de Panne
Department of Computer Science
University of British Columbia
van@cs.ubc.ca

ABSTRACT

We explore the potential of learned autocompletion methods for synthesizing animated motions from input keyframes. Our model uses an autoregressive two-layer recurrent neural network that is conditioned on target keyframes. The model is trained on the motion characteristics of example motions and sampled keyframes from those motions. Given a set of desired key frames, the trained model is then capable of generating motion sequences that interpolate the keyframes while following the style of the examples observed in the training corpus. We demonstrate our method on a hopping lamp, using a diverse set of hops from a physics-based model as training data. The model can then synthesize new hops based on a diverse range of keyframes. We discuss the strengths and weaknesses of this type of approach in some detail.

CCS CONCEPTS

• **Computing methodologies** → *Procedural animation*;

KEYWORDS

keyframe animation, neural networks, motion completion

ACM Reference Format:
Xinyi Zhang and Michiel van de Panne. 2018. Data-driven Autocompletion for Keyframe Animation. In *MIG '18: Motion, Interaction and Games (MIG '18), November 8–10, 2018, Limassol, Cyprus.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3274247.3274502

1 INTRODUCTION

Animation is a beautiful art form that has evolved artistically and technically for over many years. However, despite advancements in tools and technologies, one of the main methods used to produce animation, keyframing, remains a labor-intensive and challenging process. Artists still spend many hours posing and defining key frames or poses to choreograph motions, with a typical animator at Pixar producing around only 4 seconds of animation every one or two weeks. Over the years, many researchers have explored ways to help support and automate the animation process. An ideal system should support and accelerate the workflow while still allowing for precise art-direction of motions by the animator. The ubiquitous and classical solution of inbetweening keyframes using splines for each degree of freedom or animation variable (avar) allows for ample control, but these curves are agnostic with respect to what is being animated. Stated another way, the motion inbetweening process is not aware of the type of motion, nor its style or physics. As a result, this may then require additional keyframes to achieve a given result.

In this paper, we explore a data-driven method for motion-aware inbetweening, which can then support sparser keyframes and faster experimentation. Given a set of target key frames and their associated times, our method automatically synthesizes interpolating frames between keys, yielding an output motion that follows the movement patterns that it has seen during training, as well as allowing for a degree of extrapolation. An example is shown in Figure 1.

The key component of our method is an autoregressive recurrent neural network (ARNN) that is conditioned on the target keyframes. These are trained using a specialized loss function that places extra importance on the accuracy of the pose predictions made at the designated keyframe times. Given a set of example motions of a character to be animated, the ARNN learns the motion characteristics of the character, along with a blending function for key frame interpolation. As a consequence, motions produced by the ARNN match the movement characteristics of the character while further conforming to the art-direction provided by the artist via the keyframes. Additionally, the flexibility of an RNN model allows for the support of both tightly-spaced and loosely spaced keyframes. We train our network on a database of physics-based motion samples of a planar articulated hopping Luxo character, and generate novel motions that are choreographed using keyframing.

Figure 1: An inbetweened motion synthesis result from the learned autoregressive motion model.

The principle contribution of our work is a practical demonstration of how keyframe-conditioned motion models can be developed using current learning techniques, and the design of an autoregressive network architecture, loss function, and annealed training process in support of this. The overall style or type of motion favored by the model is implicitly specified by the training data used. Once trained, the directly-predictive nature of the model allows

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MIG '18, November 8–10, 2018, Limassol, Cyprus
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6015-9/18/11…$15.00
https://doi.org/10.1145/3274247.3274502

for fast motion synthesis as compared to the iterative optimization procedures used in trajectory-optimization approaches.

2 RELATED WORK

There is an abundance of related work that serves as inspiration and as building blocks for our work, including inbetweening, physics-based methods, data-driven animation methods, and machine learning methods as applied to computer animation. Below we note the most relevant work in each of these areas in turn.

2.1 Hand-drawn In-Betweening

Methods for automatically inbetweening hand-drawn key frames have a long history, dating back to Burtnyk and Wein [Burtnyk and Wein 1975]. Kort introduced an automatic system for inbetweening 2D drawings which identifies stroke correspondences between drawings and then interpolates between the matched strokes [Kort 2002]. Whited et al. [Whited et al. 2010] focus on the case in which two key frames are very similar in shape. They use a graph-based representation of strokes to find correspondences, and then perform a geometric interpolation to create natural, arc shaped motion between keys. More recently, Dalstein et al. [Dalstein et al. 2015] introduced a novel data structure, the Vector Animation Complex, to enable continuous interpolation in both space and time for 2D vector drawings. The VAC handles complex changes in drawing topology and supports a keyframing paradigm similar to that used by animators. For our work, we focus on the problem of inbetweening for characters that are modeled as articulated rigid-bodys. Significant advances have recently been made to allow animators to create fully-coordinated motions using a single stroke or *space-time curve (STC)* and optional secondary lines [Guay et al. 2015].

2.2 Physics-based Methods for Animation

Another important class of methods relies on simulation to automatically generate motions. However, although physics-based methods excel at producing realistic and plausible motions, they are inherently difficult to control. This is because the motions are defined by an initial state and the equations of motion, which are then integrated forward in time. However, keyframes require motions to reach pose objectives at specific instants of time in the future. One approach to the control problem for rigid bodies generates collections of likely trajectories for objects by varying simulation parameters and leave users to select the most desirable trajectory out of the possibilities, e.g., [Chenney and Forsyth 2000; Twigg and James 2007]. Space-time constraint methods [Barbič et al. 2009; Hildebrandt et al. 2012; Witkin and Kass 1988]] take another approach, treating animation as a trajectory optimization problem. These methods can treat the keyframe poses and timing as hard constraints and are capable of producing smooth and realistic interpolating motions. However, these methods are generally complex to implement, can be problematic to use whenever collisions are involved, and require explicitly-defined objective functions to achieve desired styles. The recent work of Bai et al. [Bai et al. 2016] combine interpolation with simulation to enable more expressive animation of non-physical actions for keyframed 2D cartoon animations. Their method focuses on animating shape deformations and is agnostic to the style and context of what is being animated, a drawback shared by physics-based methods. Our work focuses on the animation of articulated figures and the desire to achieve context-aware motion completion.

2.3 Data-driven Motion Synthesis

Techniques to synthesize new motions from an existing motion database include motion blending and motion graphs. Motion graphs organize large datasets of motion clips into a graph structure in which edges mark transitions between motion segments located at nodes. New motions can be generated by traversing walks on the graph, which can follow high-level constraints placed by the user, e.g., [Kovar et al. 2002]. Such methods cannot generalize beyond the space of motions present in the database. To create larger variations in what an be generated, motion blending may be used. Motion blending techniques can generate new motions satisfying high-level control parameters by interpolating between motion examples, e.g., [Grochow et al. 2004; Kovar and Gleicher 2004; Rose et al. 1998]. Relatedly, motion edits can also be performed using motion warping to meet specific offset requirements [Witkin and Popovic 1995]. The parameterizations of most motion blending methods do not support precise art-direction.

Subspace methods choose to identify subspace of motions, usually linear, in which to perform trajectory optimization. The work of Safonova et al. [Safonova et al. 2004] was among the first to propose this approach. Min et al. [Min et al. 2009] construct a space-time parameterized model which enables users to make sequential edits to the timing or kinematics. Their system models the timing and kinematic components of the motions separately and fails to capture spatio-temporal correlations. Additionally, their system is based on a linear (PCA) analysis of the motions that have been put into correspondence with each other, whereas our proposed method builds on an autoregressive predictive model that can in principle be more flexible and therefore more general. In related work, Wei et al. [Wei et al. 2011] develop a generative model for human motion that exploits both physics-based priors and statistical priors based on GPLVMs. As with [Min et al. 2009], this model uses a trajectory optimization framework.

2.4 Deep Learning for Motion Synthesis

Recently, researchers have begun to exploit deep learning algorithms for motion synthesis. One such class of methods uses recurrent neural networks (RNNs), which can model temporal dependencies between data points. Fragkiadaki et al. [Fragkiadaki et al. 2015] used an Encoder-Recurrent-Decoder network for motion prediction and mocap data generation. Crnkovic-Friis [Crnkovic-Friis and Crnkovic-Friis 2016] use an RNN to generate dance choreography with globally consistent style and composition. However, the above methods provide no way of exerting control over the generated motions. Additionally, since RNNs can only generate data points similar to those seen in the training data set in a forward manner, without modification, they cannot be used for in-filling. Recent work by Holden et al. [Holden et al. 2017, 2016] takes a step towards *controllable* motion generation using deep neural networks. In one approach [Holden et al. 2016], they construct a manifold of human motion using a convolutional autoencoder trained on a motion capture database, and then train another neural network

to map high-level control parameters to motions on the manifold. In [Holden et al. 2017], the same authors take a more direct approach for real-time controllable motion generation and train a phase-functioned neural network to directly map keyboard controls to output motions. Since [Holden et al. 2016] synthesizes entire motion sequences in parallel using a CNN, their method is better suited for motion editing, and does not support more precise control through keyframing. The method proposed in [Holden et al. 2017] generates motion in the forward direction only and requires the desired global trajectory for every time step. The promise (and challenge) of developing stable autoregressive models of motion is outlined in a number of recent papers, e.g., [Holden et al. 2016; Li et al. 2017]. The early NeuroAnimator work [Grzeszczuk et al. 1998] was one of the first to explore learning control policies for physics-based animation. Recent methods also demonstrate the feasibility of reinforcement learning as applied to physics-based models, e.g., [Peng et al. 2018, 2017] . These do not provide the flexibility and control of keyframing, however, and they currently work strictly within the space of physics-based simulations.

3 METHOD OVERVIEW

The overview of our system is represented in Figure 2. First, to create our dataset for training, we use simulation to generate a set of jumping motions for our linkage-based Luxo lamp character (§4) and preprocess the simulation data (§5) to extract sequences of animation frames along with "virtual" keyframes and timing information for each sequence. During training, we feed the ARNN network keyframes of sequences and drive the network to learn to reproduce the corresponding animation sequence using a custom loss function (§6,§7). Once the network is trained, users can synthesize new animations by providing the network with a sequence of input keyframes.

The structure of our ARNN neural network is illustrated in Figure 6 and described in greater detail later §6. The ARNN is a neural network composed of a recurrent portion and feedforward portion. The recurrent portion in conjunction with the the feedforward portion helps the net learn both the motion characteristics of the training data keyframe constraints. The ARNN takes as input a sequence of n key frames $X = \{X_0, X_1, ..., X_n\}$, along with timing information describing the temporal location of keys, $T = \{T_0, T_1, ..., T_n\}$, and outputs a final interpolating sequence of frames $Y = \{Y_0, Y_1, ..., Y_m\}$ of length $m = T_n - T_0 + 1$. Frames are pose representations, where each component X^i or Y^i describes the scalar value of a degree of freedom of Luxo.

4 THE ANIMATION DATABASE

Our animation database for training consists of hopping motions of a 2D Luxo character. In this section, we describe our physics-based method for generating motion samples for Luxo and our procedure for transforming the motion data into a format suitable for training.

4.1 Physics-based Method for Generating Animations

In order to efficiently train an expressive network, we need a sufficiently-sized motion database containing a variety of jumping motions. Creating such a dataset of jumps by hand would be impractical, so we developed a physics-based solution to generate our motion samples. We build a physics-based model of the Luxo character in Bullet with actuated joints, and use simulated annealing to search for control policies that make Luxo hop off the ground.

4.2 The Articulated Lamp Model

Figure 3 shows the mechanical configuration of Luxo which we use for simulation. The model is composed of 4 links and has 6 degrees of freedom: the x position of the base link (L1), the y position of the base link, the orientation of the base link θ_1, the joint angle θ_1 between the base link ad the the leg link (L2), the joint angle θ_2 between the leg link ad the the neck link (L3), and the joint angle θ_3 at the lamp head (L4). Despite its simple construction, the primitive Luxo is expressive and capable of a rich range of movements. To drive the motion of the character, we equip each joint with a proportional-derivative (PD) controller. The PD controller computes an actuating torque τ that moves the link towards a given target pose θ_d according to $\tau = k_p(\theta_d - \theta) - k_d\omega$ where θ and ω denote the current link position and velocity, and k and ω are stiffness and damping parameters for the controller. By finding suitable pose targets for the PD controllers over time, we can drive the lamp to jump. However, searching for admissible control policies can be intractable due to the large space of possible solutions. Thus, we restrict the policy search space by developing a simple control scheme for hops which makes the optimization problem easier.

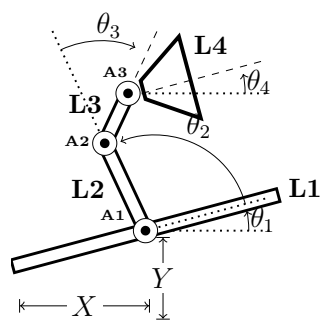

Figure 3: The mechanical configuration of Luxo

4.3 Control Scheme for Jumping

Our control scheme for Luxo, shown in Figure 4, is based on the periodic controller synthesis technique presented in [van de Panne et al. 1994]. It consists of a simple finite state machine (FSM) with a cyclic sequence of timed state transitions.The FSM behavior is governed by a set of transition duration parameters, and the target poses for each state, which then dictate the amount of torque applied to Luxo's joints, via PD controllers, when in a given state.

Given this control scheme, we can find parameter values for the transition durations and pose targets that propel Luxo forward in a potentially rich variety of hop styles. To this end, we use simulated annealing, which finds good motions by iteratively searching the parameter space using forward simulation.

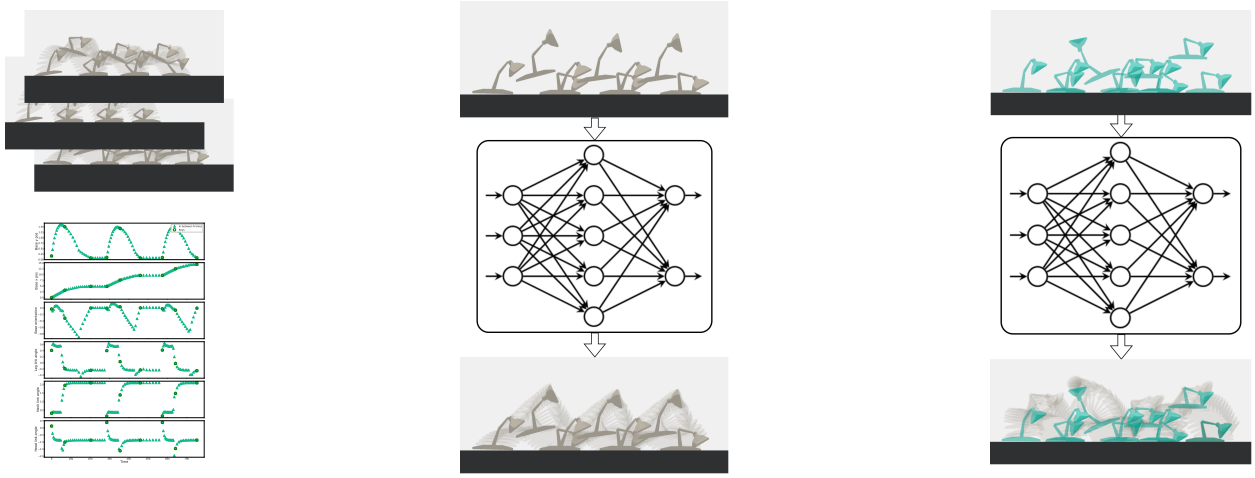

Figure 2: Overview of our system. We create our database of animations using simulation and extract keyframed animation data for training. Next, we train our ARNN neural network using back-propagation such that we reproduce the full animation sequence from input keyframes. Finally, the network can be used at run-time to generate believable character motions from new keyframe inputs.

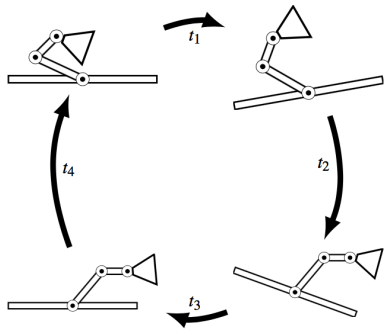

Figure 4: Pose control graph for jumping Luxo. The control graph is parameterized by the target poses for the 4 states and the transition durations t_1,t_2,t_3,t_4.

Algorithm 1 Simulated Annealing

```
function SimulatedAnnealing()
    T ← T_max
    K ← K_max
    s ← INIT()
    while K > K_max do
        s' ← NEIGHBOUR(s)
        ΔE ← E(s') − E(s)
        if RANDOM() < ACCEPT(T, ΔE) then
            s ← s'
        end if
        T ← COOLING(K)
    end while
    return s
end function
```

4.4 Finding Motions with Simulated Annealing

Algorithm 1. contains the pseudocode for the simulated annealing algorithm. The algorithm starts with an initial guess for the transition durations and target poses. At each step, the algorithm seeks to improve upon the current best guess s for the control parameters by selecting a candidate solution s' and comparing the quality, or energy of the resulting motions. If the s' produces a better jump with lower energy, the algorithm updates the current best guess. Otherwise, the algorithm probabilistically decides whether to keep the current guess according to a temperature parameter, T. At the beginning, T is set to be high to encourage more exploration of unfavorable configurations, and slowly tapered off until only a strictly better solutions are accepted.

4.4.1 The Energy Function. The quality of motions is measured by the energy function

$$\mathbf{E}(s) = -(1.0 - w_e)D - w_e H_{max}. \tag{1}$$

This function is a weighted sum of the total distance D and maximum height H_{max} reached by Luxo after cycling through the pose control graph three times using a particular policy, corresponding to three hops. To obtain a greater variety of hops, we select a random value for w_e between 0.0 and 1.0 for each run of the simulated annealing algorithm and biasing the search towards different trajectories. After K_{max} iterations of search, we record the trajectory of the best jump found if it is satisfactory, i.e reaches a certain distance and max height, and contains three hops.

4.4.2 Picking Neighboring Candidate Solutions s'. At each iteration of the simulated annealing algorithm, we choose a new candidate solution by slightly perturbing the current best state s. We select one of the 4 transition durations to perturb by 7 ms and one joint angle for each of the 4 target poses to perturb by 0.5 radians.

4.4.3 Temperature And Acceptance Probability. The probability of transitioning from the current state s to the new candidate state s' for each iteration of search is specified by the following function:

$$\mathbf{ACCEPT}(T, \mathbf{E}(s), \mathbf{E}(s')) = \begin{cases} 1, & \text{if } \mathbf{E}(s') < \mathbf{E}(s') \\ \exp(\frac{-(\mathbf{E}(s')-\mathbf{E}(s))}{T}), & \text{otherwise} \end{cases} \quad (2)$$

Thus, we transition to the new candidate state if it has strictly lower energy than the current state s'. Otherwise, we decide probabilistically whether we should explore the candidate state. From (2), we see that this probability is higher for states with lower energy and for higher temperature values T. As T cools down to 0, the algorithm increasingly favors transitions that move towards lower energy states.

4.4.4 Implementation Details. We used the Bullet physics engine [Bai and Coumans 2017] to control and simulate the Luxo character. We use a temperature cooling schedule of $T(K) = 2 * 0.999307^K$ and search for $K_{max} = 1000$ iterations for our implementation of simulated annealing. After around 10000 total runs of the algorithm, we obtained 300 different successful examples of repeatable hops for our final motion dataset. For each type we use three successive hops for training.

5 DATA PREPROCESSING

The raw simulated data generated from the above procedure consists of densely sampled pose information for jumps and must be preprocessed into a suitable format that includes plausible keyframes for training. For hops, we use the liftoff pose, the landing pose, and any pose with Luxo in the air as the 3 key poses for each jump action. To create a larger variety of motions in the training set, we also randomly choose to delete 0-2 arbitrary keys from each jump sequence every time it is fed into the network during training, so the actual training data set is much larger.

In order to extract the above key poses along with in-between frames, we first identify events corresponding to jump liftoffs and landings in the raw data. Once the individual jump segments are located, we evenly sample each jump segment, beginning at the liftoff point and ending at the landing point, to obtain 25 frames for each hop. We then take one of those 25 frames with Luxo in the air to be another keyframe, along with its timing relative to the previous keyframe. Although individual jumps may have different durations, we choose to work with the relative timing of poses within jumps rather than the absolute timing of poses for our task, and thus sample evenly within each hop segment. In the pause between landings and liftoffs, we take another 6 samples in-between frames to be included in the final sequence. The fully processed data consists of a list of key frames $X = \{X_0, X_1, ..., X_n\}$ and their temporal locations, $T = \{T_0, T_1, ..., T_n\}$, and the full sequence including key frames and in-between frames $Y = \{Y_0, Y_1, ..., Y_m\}$, where $m = T_n - T_0 + 1$. Figure 5 shows a preprocessed jump sequence from our training set.

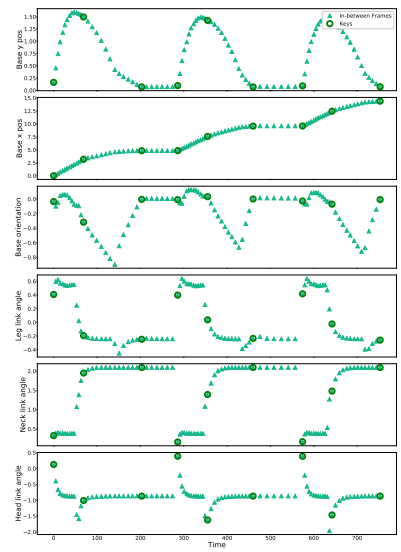

Figure 5: Preprocessed training data with extracted keyframes for each degree of freedom. There are 25 frames of animation for each full jump, punctuated by 6 frames of still transition between jumps. The bold open circles indicate extracted key frames.

6 ARNN NETWORK

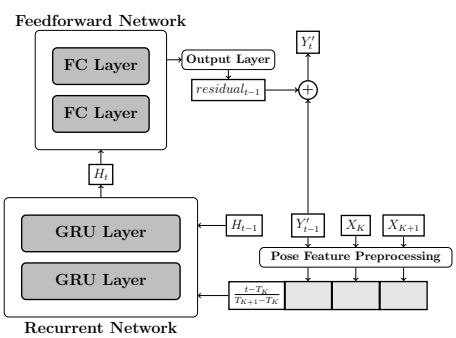

Figure 6: Architecture of the ARNN. The ARNN is composed of a recurrent portion and a feed-forward portion.

Given a sequence of key poses X and timing information T , the task of the ARNN is to progressively predict the full sequence of frames Y that interpolate between keys. The network makes its predictions sequentially, making one frame prediction at a time. To make a pose prediction for a frame t, temporally located in the interval between key poses X_K and X_{K+1}, the network takes as input the keys X_K and X_{K+1}, the previous predicted pose Y'_{t-1}, the previous recurrent hidden state H_{t-1}, and the relative temporal location

of t, which is defined according to $t_{rel} = \frac{t-T_K}{T_{K+1}-T_K}$, where T_K and T_{K+1} are the temporal locations of X_K and X_{K+1} respectively. Note that we use absolute positions for the x and y locations of Luxo's base, rather than using the relative ΔX and Δy translations with respect to the base position in the previous frame. This does not yield the desired invariance with respect to horizontal or vertical positions. However, our experiments when using absolute positions produced smoother and more desirable results than when using relative positions.

We now further describe the details of the ARNN network structure, which is composed of a recurrent and a feedforward portion, and then discuss the loss function and procedure we use to train the network to accomplish our objectives.

In the first stage of prediction, the network takes in all the relevant inputs including the previous predicted frame Y'_{t-1}, the previous hidden state H_{t-1}, the preceding key frame X_K, the next key frame X_{K+1}, and the relative temporal location of t, t_{rel} to produce a new hidden state H_t.

Y'_{t-1}, X_K, and X_{K+1} are first individually preprocessed by a feedforward network composed of two linear layers as a pose feature extraction step before they are concatenated along with t_{rel} to form the full input feature vector at time t, x_t. This concatenated feature vector along with the hidden state H_{t-1} is fed into the recurrent portion of the network, composed of two layers of GRUs [Cho et al. 2014]. We use SELU activations [Klambauer et al. 2017] between intermediate outputs. The GRU cells have 100 hidden units each, with architectures described by the following equations:

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}H_{(t-1)} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}H_{(t-1)} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t(W_{hn}H_{(t-1)} + b_{hn})) \\ H_t &= (1 - z_t)n_t + z_tH_{(t-1)} \end{aligned} \tag{3}$$

For the second stage of the prediction, the network takes the hidden output from the previous stage H_t as input into a feedforward network composed of two linear layers containing 10 hidden units each. This output from the feedforward network is then mapped to a 6 dimensional residual pose vector which is added to the previous pose prediction Y'_{t-1} to produce the final output Y'_t.

The recurrent and feedforward portions of the network work together to accumulate knowledge about keyframe constraints while learning to make predictions that are compatible with the history of observed outputs. This dual-network structure arose from experiments showing that a RNN only network/a feedforward only network is insufficient for meeting the competing requirements of self-consistency with the motion history and consistency with the keyframe constraints. Our experiments with other architectures are detailed in §9.

7 TRAINING

7.1 Loss Function

During the training process, we use backpropagation to optimize the weights of the ARNN network so that the net can reproduce the full sequence of frames Y given the input keyframe information X and T. Additionally, to drive the network to learn to interpolate between keyframe constraints, we use a specialized loss function,

$$\begin{aligned} L_{ARNN} &= 100\omega \sum_{K=1}^{n} (X_K - Y'_{T_K})^2 + MSE(Y, \hat{Y}) \\ MSE(W, Z) &= \frac{1}{N}\sum_{i=1}^{N}(W_i - Z_i)^2. \end{aligned} \tag{4}$$

This custom loss function is composed of two parts - the frame prediction loss and the key loss. The frame prediction loss, $MSE(Y, Y')$, is the vanilla mean-squared error loss which calculates the cumulative pose error between poses in the final predicted sequence and the ground truth sequence. This loss helps the network encode movement information about the character during training as it is coerced to reproduce the original motion samples. By itself this frame loss is insufficient for our in-betweening task because it fails to model the influence of keyframe constraints on the final motion sequence. In order to be consistent with the keyframing paradigm, in-between frames generated by the network should be informed by the art-direction of the input keys and interpolate between them. Consequently, we introduce an additional loss term to the total loss - the key loss, $\sum_{K=1}^{n}(X_K - Y'_{t_K})^2$ to penalize discrepancies between predicted and ground truth *keys*, forcing the network to pay attention to the input keyframe constraints. Amplifying the weight of this loss term simulates placing a hard constraint on the network to always produce frame predictions that hit the original input keyframes. In experiments, we found that a weight of 100 was sufficient. However, because the network must incorporate the contrasting demands of learning both the motion pattern of the data as well as keyframe interpolation during training, setting the weight of the key loss to 100 in the beginning is not optimal. Consequently, we introduce ω, the *Key Importance Weight*, which we anneal during the training process as part of a curriculum learning method [Bengio et al. 2009] to help the network learn better during training.

7.2 Curriculum Learning

In our curriculum learning method, we set ω to be 0 in the beginning stages of training and slowly increase ω during the latter half of the training process. Thus, for the first part of the training process the network primarily focuses on learning the motion characteristics of the data. Once the first stage of learning has stabilized, we increase ω so the network can begin to consider the keyframe constraints.During the first phase of the training, we apply scheduled sampling [Bengio et al. 2015] as another curriculum learning regime to help the recurrent portion of the net learn movement patterns. In this scheme, we feed the recurrent network the ground truth input for the previous predicted pose Y_{t-1} instead of the recurrent prediction Y'_{t-1} at the start of training, and gradually change the training process to fully use generated predictions Y'_{t-1}, which corresponds to the inference situation at test time. The probability of using the ground truth input at each training epoch is controlled by the teacher forcing ratio which is annealing using an inverse sigmoid schedule shown by the green curve of Figure 7. Once the learning has stabilized, we gradually increase ω and push the network to learn the interpolation aspect of the prediction until the network is able to successfully make predictions for the training

samples under the new loss function. The sigmoid decay schedule for ω is shown by the blue curve in Figure 7.

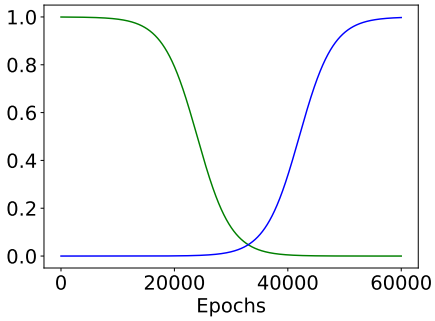

Figure 7: Curriculum learning schedules used to train the ARNN network during 60000 training epochs: teacher forcing ratio decay (decreasing; green) and key importance weight ω annealing (increasing; blue)

Without curriculum learning, the network has a harder time learning during the training process and we observe a large error spike in the learning curve at the beginning of training. In contrast, using the above curriculum learning method produces a smoother learning curve and superior results. Table 1 shows the quantitative loss achieved on the test set for the ARNN trained with curriculum learning and without curriculum learning.

Table 1: The ARNN trained with vs without curriculum on a smaller sample set of 80 jump sequences for 20000 epochs. Curriculum learning results lower loss values.

Training	Key Loss	Frame Loss	Total Loss
With ω annealing	0.00100	0.00395	0.00496
No ω annealing	0.00120	0.00670	0.00790

7.3 Training Details

The final model we used to produce motions in the results section is trained on 240 jump sequences, with 3 jumps per sequence. As noted above, from each jump sequence we obtain many more sequences by randomly removing 0, 1, or 2 arbitrary keys from the sequence each time it is fed into the network. The model is optimized for 60000 epochs using Adam [Kingma and Ba 2014] with an initial learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ and regularized using dropout [Srivastava et al. 2014] with probabilities 0.9 and 0.95 for the first and second GRU layers in the recurrent portion of the network and probabilities 0.8 and 0.9 for the first and second linear layers in the feedforward portion. The current training process takes 130 hours on an NVIDIA GTX 1080 GPU, although we expect that significant further optimizations are possible.

8 RESULTS

We demonstrate the autocompletion method by choreographing novel jumping motions using our system. When given keyframe constraints that are similar to those in the training set, our system reproduces the motions accurately. For keyframe constraints that deviate from those seen in the training dataset, our system generalizes, synthesizing smooth motions even with keyframe inputs that are physically unfeasible. Results are best seen in the accompanying video.

We first test our system's ability to reproduce motions from the test dataset, i.e., motions that are excluded from the training data, based on keyframes derived from those motions. The animation variables (AVARs) for two synthesized jumps from the test set are plotted in Figure 8. More motions can be seen in the video. The trajectories generated using our system follow the original motions closely, and accurately track almost all of the keyframe AVARs. The motions output by our system are slightly smoother than the original jumps, possibly due to predictions regressing to an average or the residual nature of the network predictions.

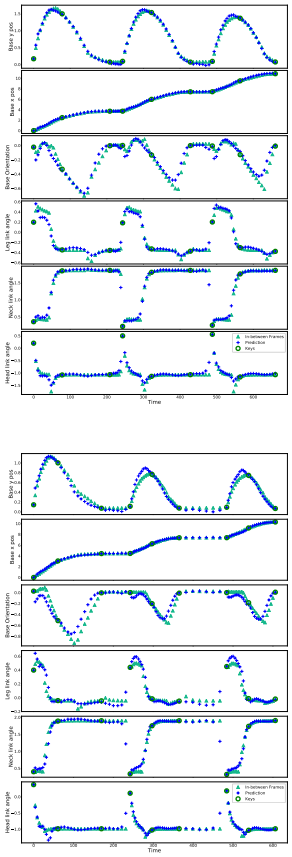

Figure 8: Motion reconstruction of two jumps taken from the test set. AVARs from the original simulated motion trajectory from the test set is plotted in light green. Extracted keyframes are circled in dark green. The resulting motion generated using our network is displayed in dark blue.

We next demonstrate generalization by applying edits of increasing amplitudes to motions in the test set. Our system produces

plausible interpolated motions as we modify keyframes to demand motions that are increasingly non-physical and divergent from the training sets. Figure 9 shows a sequence of height edits to a jump taken from the test set. As we decrease or increase the height of the keyframe at the apex, the generated motion follows the movements of the original jump and tracks the new keyframe constraints with a smooth trajectory. Next, we modify the timing of another sequence from the test set. Timing edits are shown in Figure 10. Here, our system generates reasonable results but they do not track the keyframe constraints as precisely; in the second figure the new timing requires that Luxo cover the first portion of the jump in half the original amount of time. This constraint deviates far from motions of the training set, and motion generated by the network is biased toward a more physically plausible result.

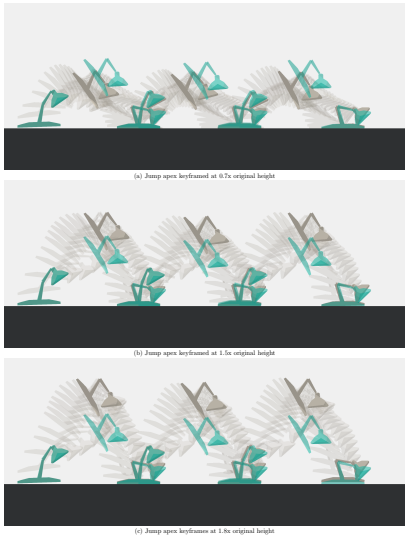
(a) Jump apex keyframed at 0.7x original height
(b) Jump apex keyframed at 1.5x original height
(c) Jump apex keyframes at 1.8x original height

Figure 9: Height edits. Original extracted keyframes are shown in green. Generated motions track new apex keyframes that have heights of 0.7× (top), 1.5× (middle), and 1.8× (bottom) the original base height.

We next show that in the absence of keyframes, i.e, with sparse key inputs, the network is still able to output believable trajectories, demonstrating that the predictions output by the network are style-and-physics aware. In Figure 11, we have removed the apex keyframe and the landing keyframe of the second jump in the sequence. Our network creates smooth jump trajectories following the movement characteristics of the Luxo character despite the lack of information. This enabled by the memory embodied in the recurrent model, which allows the network to build an internal motion model of the character to make rich predictions.

The non-linearity and complexity of the motions as output by the network can also be see in Figure 12. Figure 12 shows the changes of individual degrees of freedom resulting from an edit to the height of the keyframe, as seen in the top graph. If this edit were made with a simple motion warping model, applied individually to each degree of freedom, then we would expect to see an offset in the top graph that is slowly blended in and then blended out. Indeed, for the base position, the motion deformation follows a profile of approximately that nature. However, the curves for the remaining degrees of freedom are also impacted by the edit, revealing the coupled nature of the motion synthesis model.

A number of results are further demonstrated by randomly sampling, mixing, and perturbing keyframes from the test dataset, as shown in Figure 13.

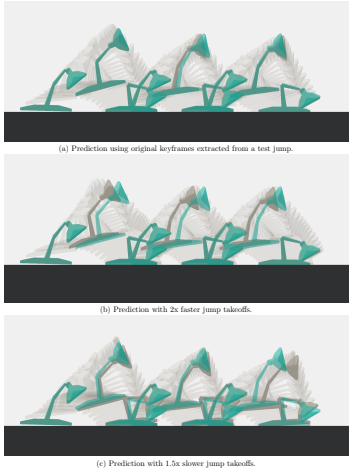
(a) Prediction using original keyframes extracted from a test jump.
(b) Prediction with 2x faster jump takeoffs.
(c) Prediction with 1.5x slower jump takeoffs.

Figure 10: Timing edits. Original extracted keyframes are shown in green. The predicted pose at key locations are shown in dark gray. (top) Prediction using original keyframes, with keyframe numbers n=(13,46,79). (middle) Keyed for faster takeoff, using n=(7,40,73). (bottom) Keyed for slower takeoff, using n=(19,52,85).

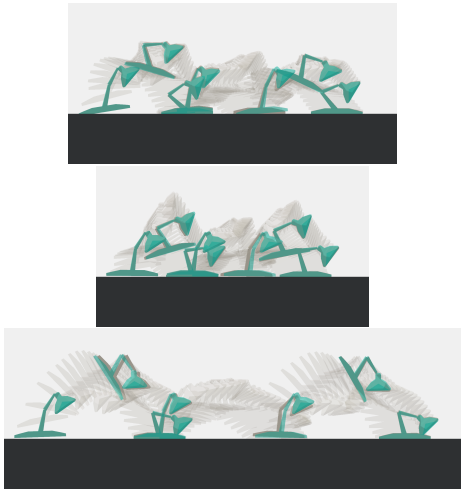

Figure 11: Motion generation with sparse keyframes. The top and landing keyframes for the above jumps have been removed from the input.

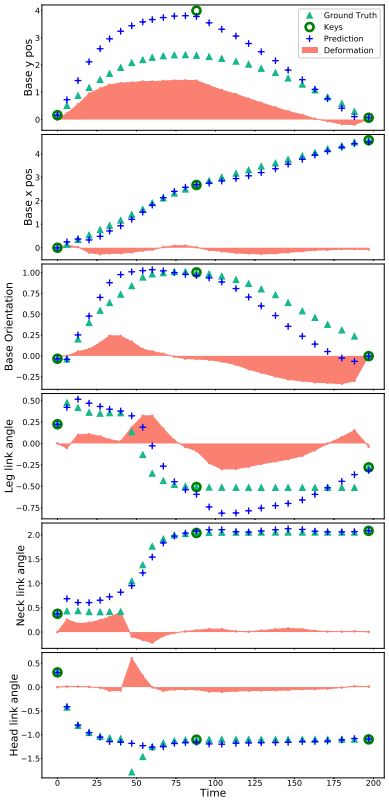

Figure 12: Coupled nature of the motion synthesis model. Edits to a single degree of freedom (top graph, base y position) leads to different warping functions for the other degrees of freedom.

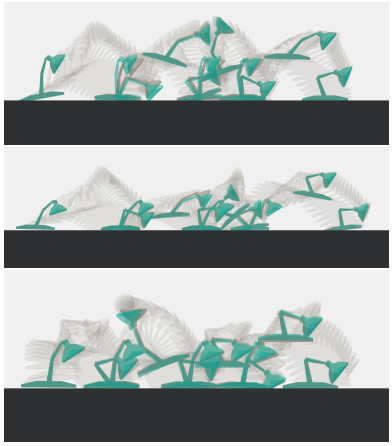

Figure 13: Motion synthesis from novel keyframe inputs. We created new keyframes from randomly sampled and perturbed keys taken from the test set (green). The output motion from the network is shown with predicted poses at input key locations shown in dark gray.

9 COMPARISON TO OTHER ARCHITECTURES

We evaluated other network architectures for the task of keyframe auto-completion including a keyframe conditioned feed-forward network with no memory and a segregated network which separates the motion pattern and interpolation portions of the task. The architecture of the segregated network is show in Figure 14. This produces a pure RNN prediction with no keyframe conditioning that is then "corrected" with a residual produced by a keyframe conditioned feed-forward network. The number of layers and hidden units for all the networks are adjusted to produce a fair comparison and all networks are trained on 80 sample jump sequences until convergence.

Table 2: The test losses from other network architectures vs the ARNN. The ARNN produces the best overall losses.

Architecture	Key Loss	Frame Loss	Total Loss
ARNN	0.00100	0.00396	0.00496
No Memory Net	0.00018	0.00817	0.00835
Segregated Net	0.00124	0.01609	0.01891

Quantitatively, the ARNN produces the lowest total loss out of the three architectures (Table 2) and the segregated net produces the worst loss. The ARNN also produces the best results qualitatively, while the feed-forward only network produces the least-desirable results due to motion discontinuities. Figure 15 shows the failure case when a network has no memory component. Although the feed-forward only network generates interpolating inbetweens that are consistent with one another *locally* within individual motion segments, *globally* across multiple keys, the inbetweens are not coordinated, leading to motion discontinuities. This most evident in the predictions for the base x AVAR. The segregated net and the ARNN both have memory, which allows these nets to produce smoother predictions with global consistency. Ultimately however, the combined memory and keyframe conditioning structure of the ARNN produces better results than the segregated net which separates the predictions.

10 CONCLUSIONS

In this paper, we explored a conditional autoregressive method for motion-aware keyframe completion. Our method synthesizes motions adhering to the art-direction of input keyframes while following the style of samples from the training data, combining intelligent automation with flexible controllability to support and accelerate the animation process. In our examples, the training data comes from physics-based simulations, and the model produces plausible reconstructions when given physically-plausible keyframes. For motions that are non-physical, our model is capable of generalizing to produce smooth motions that adapt to the given keyframe constraints.

The construction of autoregressive models allows for a single model to be learned for a large variety of character movements. Endowed with memory, our network can learn an internal model of the movement patterns of the character and use this knowledge to intelligently extrapolate frames when in the absence of from

keyframe guidance. The recurrent nature of our model allows it to operate in a fashion that is more akin to a simulation, i.e., it is making forward predictions based on a current state. This has advantages, i.e., simplicity and usability in online situations, and disadvantages, e.g., lack of consideration of more than one keyframe in advance, as compared to motion trajectory optimization methods. As noted in previous work, the construction of predictive autoregressive models can be challenging, and thus the proposed conditional model is a further proof of feasibility for this class of model and its applications.

Trajectory optimization methods are different in nature to our work, as they require require an explicit model of the physics and the motion style objectives. In contrast, autoregressive models such as ours make use of a data-driven implicit model of the dynamics that encompasses the motion characteristics of the example motions. These differences make a meaningful direct comparison difficult. The implicit modeling embodied by the data-driven approach offers convenience and simplicity, although this comes at the expense of needing a sufficient number (and coverage) of motion examples. Our work also shares some of the limitations of trajectory-based optimization methods, such as potential difficulties in dealing with collisions. However, if a physics-based solution is realizable with the given keyframe constraints, the iterative nature of a trajectory optimization method may in practice yield movements with a lower residual physics error, if any.

The method we present, together with its evaluation, still has numerous limitations. The method is dependent on the generation of an appropriate database of motions to use for training. This can be a complex and challenging problem by itself, although we believe that a vary of retargeting and data augmentation methods can potentially help here. For example, we wish to further improve the motion coverage of our example data by collecting physics-based motions in reduced gravity environments. If target keyframes go well beyond what was seen in the training set, the motion quality may suffer. While the current results represents an initial validation of our approach, we wish to apply our model to more complex motions and characters; it may or may not scale to these cases. A last general problem with the type of learning method we employ is that of reversion to the mean when there are multiple valid possibilities for reaching a given keyframe. In future work, we wish to develop methods that can sample from trajectory distributions.

Currently a primary extrapolation artifact is an apparent loss of motion continuity in the vicinity of the keyframes, which can happen when the model generates in-betweens that fail to interpolate the keyframes closely. This artifact could likely be ameliorated with additional training that further weights the frame prediction loss after ω has been annealed. This could help the network consolidate knowledge and produce smoother results. Discontinuities in motions caused by collision impulses are also not fully resolved in our method. These are modeled implicitly in the learned model and the resulting motion quality suffers slightly as a result. An alternative approach would be to add explicit structure to the learned model in support of modeling collision events. Lastly, the current model we have developed does not provide partial keyframe control; the full set of AVAR values for the character must be specified per keyframe. This is in part due to the nature of the artificial training data set we've created for this paper, which does not include partial keyframes. We would like to add support for partial keyframing in the future and test the applicability of our system for production use on real animation data sets.

REFERENCES

Yunfei Bai and Erwin Coumans. 2016-2017. a python module for physics simulation in robotics, games and machine learning. http://pybullet.org/.

Yunfei Bai, Danny M. Kaufman, C. Karen Liu, and Jovan Popović. 2016. Artist-directed Dynamics for 2D Animation. *ACM Trans. Graph.* 35, 4, Article 145 (July 2016), 10 pages. https://doi.org/10.1145/2897824.2925884

Jernej Barbič, Marco da Silva, and Jovan Popović. 2009. Deformable Object Animation Using Reduced Optimal Control. In *ACM SIGGRAPH 2009 Papers (SIGGRAPH '09)*. ACM, New York, NY, USA, Article 53, 9 pages. https://doi.org/10.1145/1576246.1531359

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15)*. MIT Press, Cambridge, MA, USA, 1171–1179. http://dl.acm.org/citation.cfm?id=2969239.2969370

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*. ACM, New York, NY, USA, 41–48. https://doi.org/10.1145/1553374.1553380

Nestor Burtnyk and Marceli Wein. 1975. Computer animation of free form images. In *ACM SIGGRAPH Computer Graphics*, Vol. 9. ACM, 78–80.

Stephen Chenney and D. A. Forsyth. 2000. Sampling Plausible Solutions to Multi-body Constraint Problems. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 219–228. https://doi.org/10.1145/344779.344882

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR* abs/1409.1259 (2014). arXiv:1409.1259 http://arxiv.org/abs/1409.1259

Luka Crnkovic-Friis and Louise Crnkovic-Friis. 2016. Generative Choreography using Deep Learning. *CoRR* abs/1605.06921 (2016). http://arxiv.org/abs/1605.06921

Boris Dalstein, Rémi Ronfard, and Michiel van de Panne. 2015. Vector Graphics Animation with Time-varying Topology. *ACM Trans. Graph.* 34, 4, Article 145 (July 2015), 12 pages. https://doi.org/10.1145/2766913

Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. 2015. Recurrent Network Models for Kinematic Tracking. *CoRR* abs/1508.00271 (2015). http://arxiv.org/abs/1508.00271

Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-based Inverse Kinematics. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 522–531. https://doi.org/10.1145/1015706.1015755

Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. 1998. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 9–20.

Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani. 2015. Space-time sketching of character animation. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 118.

Klaus Hildebrandt, Christian Schulz, Christoph von Tycowicz, and Konrad Polthier. 2012. Interactive Spacetime Control of Deformable Objects. *ACM Trans. Graph.* 31, 4, Article 71 (July 2012), 8 pages. https://doi.org/10.1145/2185520.2185567

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (July 2017), 13 pages. 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. Graph.* 35, 4, Article 138 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925975

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-Normalizing Neural Networks. *CoRR* abs/1706.02515 (2017). arXiv:1706.02515 http://arxiv.org/abs/1706.02515

Alexander Kort. 2002. Computer Aided Inbetweening. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering (NPAR '02)*. ACM, New York, NY, USA, 125–132. https://doi.org/10.1145/508530.508552

Lucas Kovar and Michael Gleicher. 2004. Automated Extraction and Parameterization of Motions in Large Data Sets. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 559–568. https://doi.org/10.1145/1015706.1015760

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. *ACM Trans. Graph.* 21, 3 (July 2002), 473–482. https://doi.org/10.1145/566654.566605

Zimo Li, Yi Zhou, Shuangjiu Xiao, Chong He, and Hao Li. 2017. Auto-Conditioned LSTM Network for Extended Complex Human Motion Synthesis. *arXiv preprint arXiv:1707.05363* (2017).

Jianyuan Min, Yen-Lin Chen, and Jinxiang Chai. 2009. Interactive Generation of Human Animation with Deformable Motion Models. *ACM Trans. Graph.* 29, 1, Article 9 (Dec. 2009), 12 pages. https://doi.org/10.1145/1640443.1640452

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 (Proc. SIGGRAPH 2018)* 37, 4 (2018).

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)* 36, 4 (2017).

Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. 1998. Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Comput. Graph. Appl.* 18, 5 (Sept. 1998), 32–40. https://doi.org/10.1109/38.708559

Alla Safonova, Jessica K Hodgins, and Nancy S Pollard. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (ToG)* 23, 3 (2004), 514–521.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958. http://dl.acm.org/citation.cfm?id=2627435.2670313

Christopher D. Twigg and Doug L. James. 2007. Many-worlds Browsing for Control of Multibody Dynamics. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. ACM, New York, NY, USA, Article 14. https://doi.org/10.1145/1275808.1276395

Michiel van de Panne, Ryan Kim, and Eugene Flume. 1994. Virtual Wind-up Toys for Animation. In *Proceedings of Graphics Interface '94*. 208–215.

Xiaolin Wei, Jianyuan Min, and Jinxiang Chai. 2011. Physically valid statistical models for human motion generation. *ACM Transactions on Graphics (TOG)* 30, 3 (2011), 19.

B. Whited, G. Noris, M. Simmons, R. Sumner, M. Gross, and J. Rossignac. 2010. BetweenIT: An Interactive Tool for Tight Inbetweening. *Comput. Graphics Forum (Proc. Eurographics)* 29, 2 (2010), 605–614.

Andrew Witkin and Michael Kass. 1988. Spacetime Constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*. ACM, New York, NY, USA, 159–168. https://doi.org/10.1145/54852.378507

Andrew Witkin and Zoran Popovic. 1995. Motion warping. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 105–108.

A APPENDIX A

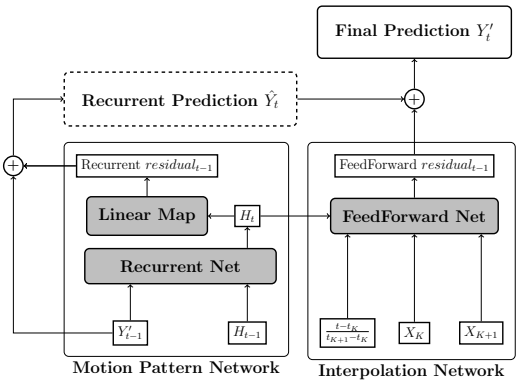

Figure 14: Architecture of the segregated network which combines a RNN only prediction produced by the Motion Pattern Network, with a keyframe conditioned "correction" produced by a feed-forward Interpolation Network.

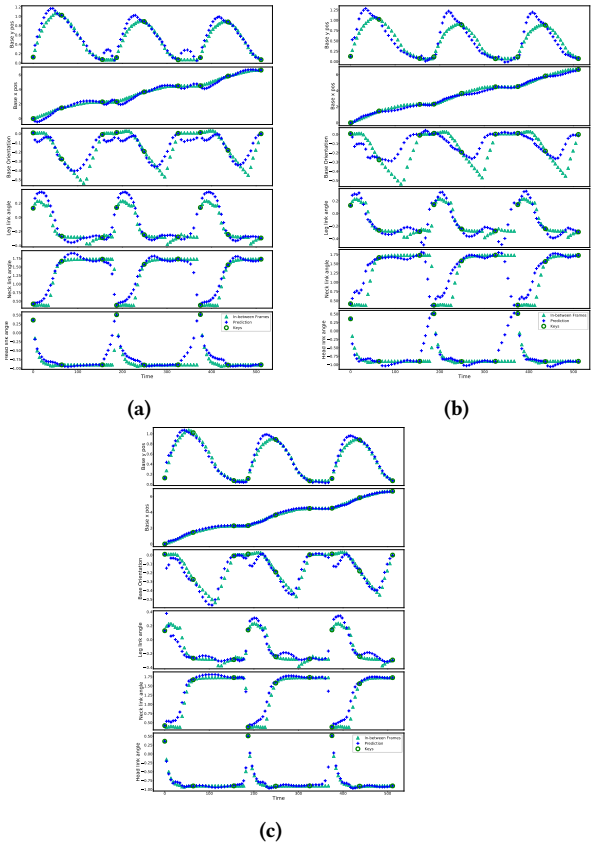

Figure 15: Qualitative comparison between the no memory net (a), segregated net (b), and the ARNN (c). The ARNN and segregated net produce smoother motions at key transitions with the use of memory.