Data Driven Auto-completion for Keyframe Animation

by

Xinyi Zhang

B.Sc, Massachusetts Institute of Technology, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

August 2018

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Data Driven Auto-completion for Keyframe Animation

submitted by **Xinyi Zhang** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Michiel van de Panne, Computer Science
Supervisor

Leonid Sigal, Computer Science
Second Reader

Abstract

Keyframing is the main method used by animators to choreograph appealing motions, but the process is tedious and labor-intensive. In this thesis, we present a data-driven autocompletion method for synthesizing animated motions from input keyframes. Our model uses an autoregressive two-layer recurrent neural network that is conditioned on target keyframes. Given a set of desired keys, the trained model is capable of generating a interpolating motion sequence that follows the style of the examples observed in the training corpus.

We apply our approach to the task of animating a hopping lamp character and produce a rich and varied set of novel hopping motions using a diverse set of hops from a physics-based model as training data. We discuss the strengths and weaknesses of this type of approach in some detail.

Lay Summary

Computer animators today use a tedious process called keyframing to make animations. In this process, animators must carefully define a large number of guiding poses, also known as keyframes for a character at different times in an action. The computer then generates smooth transitions between these poses to create the final animation. In this thesis, we develop a smart animation auto-completion system to speed up the keyframing process by making it possible for animators to define fewer keyframes. Using statistical models, our system learns a character's movement patterns from previous animation examples and then incorporates this knowledge to generate longer intervals between keyframes.

Preface

This thesis is submitted in partial fulfillment of the requirements for a Master of Science Degree in Computer Science. The entire work presented here is original work done by the author, Xinyi Zhang, performed under the supervision of Dr. Michiel Van De Panne with code contributions from Ben Ling on the 3D visualizer for displaying results. A version of this work is currently in review for the 2018 Motion, Interaction, and Games conference.

Table of Contents

List of Tables

List of Figures

Glossary

ARNN	autoregressive recurrent neural network
AVAR	animation variable
FSM	finite state machine
GRU	gated recurrent unit
RNN	recurrent neural network
SELU	scaled exponential linear unit

Acknowledgments

Firstly, I would like to thank my supervisor Dr. Michiel van de Panne for helping me turn my fanciful ideas about animation into reality. His optimism and good cheer are truly inspiring. Whenever I encountered moments of difficulty, Michiel would always patiently work with me to find ways to move forward. I am incredibly grateful to have had the opportunity to learn from his wisdom and knowledge.

I wish to thank my lab mates Glen Berseth, Boris Dalstein, Xue Bin Peng, Jacob Chen, Zhaoming Xie, and Chenxi Liu for helping me feel at home in graduate school with their companionship and support. Additionally, I extend my thanks to Dr. Leonid Sigal for his suggestions and comments on my thesis and Dr. Daniel Holden for his helpful advice on troubleshooting neural networks. I also wish to thank all of my colleagues and friends from the computer graphics community who have supported me throughout this journey.

To my parents, Hongbing Zhang and Manhua Song, and the best brother one could ever have, Daniel Zhang, thank you for always being there for me, for supporting me through the tough times, and for helping me discover the good in life. I'm forever grateful for all that you've given me and all that you continue to give me.

My time in Vancouver has been magical because of my wonderful friends. I'd never imagined I'd be snow camping, having bear adventures (I forgive you Sabnam), or wearing yoga leggings everywhere, but here I am, kind of missing the rain. A special thanks to Sabnam, Jessica Yu, and my Vancouver friends for showing me the beauty of this city. I would also like to thank my friends at home and abroad for their messages and visits, which made my days so much brighter.

Finally, all of this is really the fault of Charles Badiller and Omar Elhindi, two very talented animators I met during my internship at Disney in 2014. If I hadn't sat next to them that summer witnessing their frustrations, none of this work would have been possible. I wish to thank Charles, Omar, and artists everywhere for their dedication and passion. Your work has been the inspiration for all that I've done here.

Chapter 1

Introduction

Animation is a beautiful art form that has evolved artistically and technically 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. In this process, sets of poses, or *keyframes*, are placed at critical moments in an action, and the animation system generates smoothly transitioning *inbetweens* between keys to complete the motion. For artists, keyframing offers a low-level degree of timing and positional control that enables them to create highly expressive animations. However, the keyframing process is slow and labor-intensive. The spline curves generally used for interpolation are not aware of the type of motion, nor its style or physics. Stated another way, the motion inbetweening process is agnostic to what is being animated. Consequently, animators must often do extra work to achieve a desired result, either by manually adjusting the interpolation curve parameters or by inserting many additional keyframes. In a production environment where characters often have with hundreds of animation variables (AVARS) controlling different degrees of freedom, this way of working quickly becomes unmanageable[35]. It is not uncommon for artists to 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.

1.1 Problems and Challenges

Over the years, many researchers have sought ways to automate the animation process, but this a challenging task. To be helpful to animators, a good system would need to generate motions comparable to those produced by professionals. However, crafting high-quality motions requires tremendous expertise, and animators take into account many factors in the creation of animations, including advanced knowledge of physical laws, acting, and visual appeal. In order to achieve non-trivial levels of automation, parts of this expertise must somehow be emulated. Further, being able to generate quality motions is not enough. In order to be practically useful, an automation system should afford artists a high degree of creative freedom and facilitate artistic control. Previous approaches to the problem using physics-based or data-driven techniques to augment the animation process are seldom used in practice because they fail to support low-level timing or positional control of motions. An ideal automation system should support and accelerate the work flow while still allowing for precise art-direction of motions by artists.

1.2 Our approach

In this thesis, we address the issues discussed above by introducing a novel data-driven method for automation that supports low-level keyframe control. The core component of our method is a new autoregressive recurrent neural network (ARNN) architecture 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 style and movement characteristics of the character and conform to the art-direction provided by the artist through keyframing. Additionally, the flexibility of an recurrent neural network (RNN) model allows our method to naturally support the ability for artists to control the level of automation by applying the method to either tightly-spaced or 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 of the character choreographed using keyframing. The results demonstrate the expressibility and flexibility of our system.

1.3 Thesis Overview

The remainder of this thesis is organized as follows. In Chapter 2, we review previous approaches for automatic inbetweening and related work in motion synthesis and deep learning. Chapter 3 describes our process for constructing the animation data set we use to train the ARNN network. Chapter 4 details the network architecture and training procedure. Chapter 5 presents and discusses the results produced using our method. Lastly, in Chapter 6, we discusses the limitations of our system and possible future directions for the work.

Chapter 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 [6]. Kort introduced an automatic system for inbetweening 2D drawings which identifies stroke correspondences between drawings and then interpolates between the matched strokes [20]. Whited et al. [34] 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. [11] introduced a novel data structure, the Vector Animation Complex, shown in Figure 2.1 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 bodies.

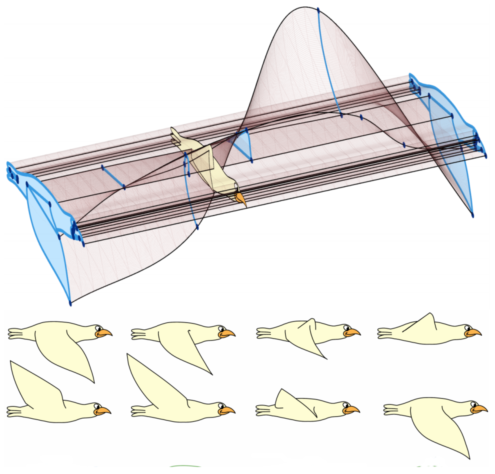


Figure 2.1: Visualization of the Vector Animation Complex [11] data structure (top) for a 2D bird animation (bottom). (Reproduced from Figure 10 of [11] with permission.)

2.2 Computer-based In-Betweening

Inbetweening in computer animation is performed semi-automatically. Animators use control points to precisely define interpolating splines which blend between parameters (i.e AVARS) at keyframes to generate smooth animations. As noted in Section 1, animators typically touch hundreds of AVARS when animating, and defining spline curves for individual AVARS tedious. Methods enabling faster edits of animation curves include motion warping [37], which preserves local movement patterns during edits by shifting and scaling original animation curves to satisfy new keyframe constraints, reducing the work required to adjust individual splines. Staggered poses [9] encode timing relationships between coordinated AVARS and preserves inter-AVAR movement patterns during edits of correlated movements. There has been less work on the *generation* of inbetweening splines. Shen et al. [29] developed a procedural method for automatically generating detailed coordinated motions using a minimum number of AVARS. However, their method is limited to mostly cyclical motions. Nebel et al. [25] generated interpolating splines from keyframes with the objective of avoiding self-collisions between the limbs of characters. Our objective in this thesis of generating appealing, style and motion aware inbetweens is more complex.

2.3 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., [7, 31]. Space-time constraint methods [3, 15, 36]] 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. [2] combine interpolation with simulation to enable more expressive animation of non-physical actions for keyframed 2D cartoon animations. Their method focuses on secondary animation of 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 for primary animation.

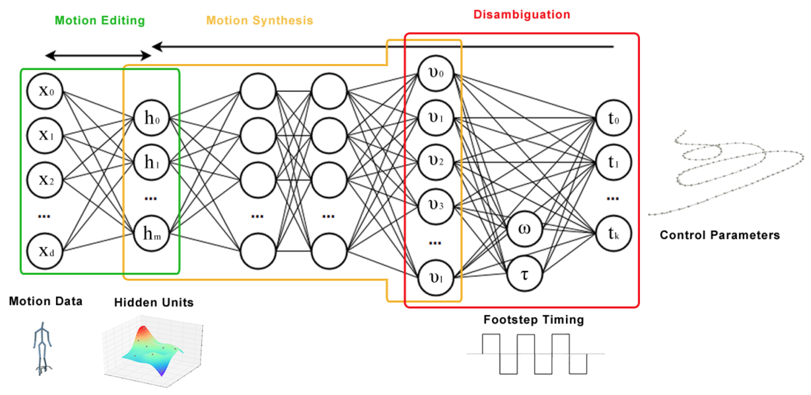
2.4 Data-driven Motion Synthesis and Animation

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., [22]. 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., [13, 21, 27]. Relatedly, motion edits can also be performed using motion warping to meet specific offset requirements [38]. The parameterizations of most motion blending methods do not support precise art-direction.

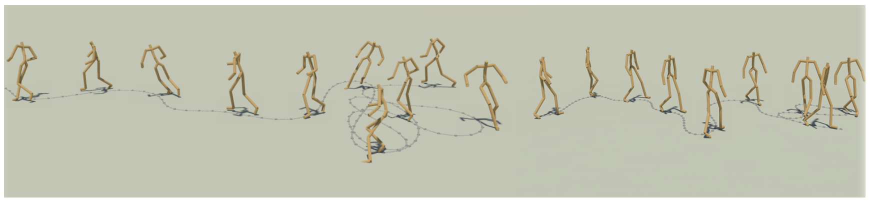
Subspace methods choose to identify subspace of motions, usually linear, in which to perform trajectory optimization. The work of Safonova et al. [28] was among the first to propose this approach. Min et al. [24] construct a space-time parameterized model which enables users to make sequential edits 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 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. [33] develop a generative model for human motion that exploits both physics-based priors and statistical priors based on GPLVMs. As with [24], this model uses a trajectory optimization framework.

2.5 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. [12] used an Encoder-Recurrent-Decoder network for motion prediction and mocap data generation. Crnkovic-Friis [10] 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. [16, 17] takes a step towards *controllable* motion generation using deep neural networks. In one approach [16] illustrated in Figure 2.2 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 [17], 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 [16] 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 [17] 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., [16, 23]. The early NeuroAnimator work [14] 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., [26]. These do not provide the flexibility and control of keyframing, however, and they currently work strictly within the space of physics-based simulations.



(a)



(b)

Figure 2.2: (a) The neural network architecture developed by Holden et al. [16] for motion synthesis. The feed forward network network maps high level control parameters to motion in the hidden space of a convolutional autoencoder. (b) Character motion generated from an input trajectory using the method developed in [16]. (Reproduced from Figure 1 and 2 of [16] with permission.)

Chapter 3

Method Overview

In this section, we describe the details of our system, which is show in Figure 3.1. First, to create our dataset for training, we use simulation to generate a set of jumping motions for our linkage-based Luxo lamp character (see Section 3.1) and preprocess the simulation data (see 3.2) 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 (see Section 3.3, Section 3.4). 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 3.6 and described in greater detail in Section 3.3. 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.

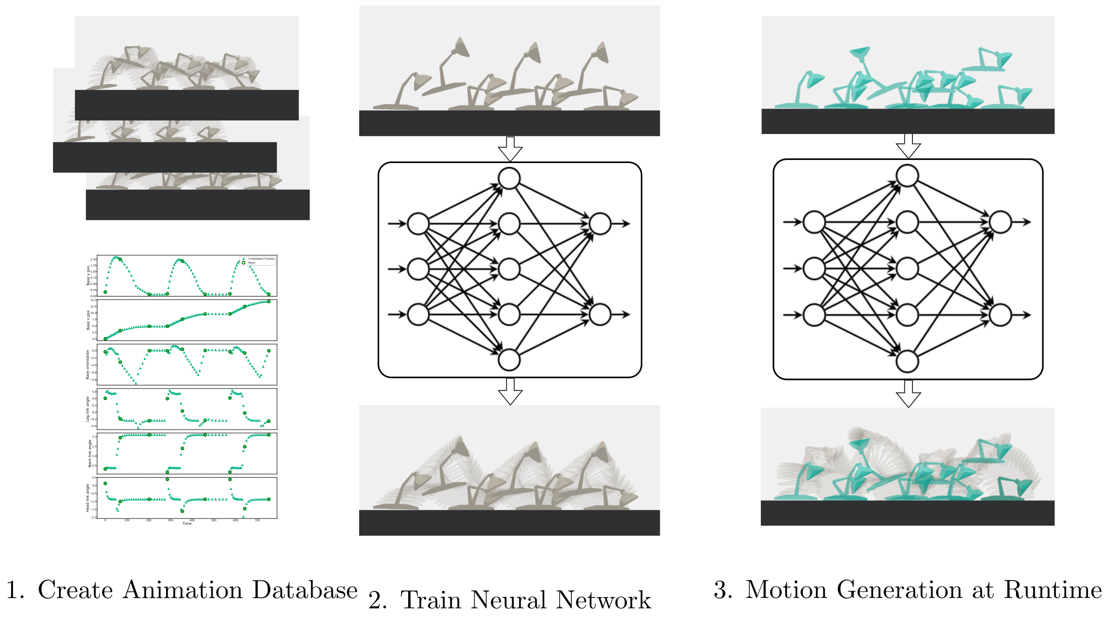


Figure 3.1: The three stages of our system: creation of the animation database, training of the ARNN network, and generation of believable motions at runtime.

3.1 The Animation Database

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. Our animation database for training consists of hopping motions of a 2D Luxo character.

3.1.1 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.

3.1.2 The Articulated Lamp Model

Figure 3.2 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.

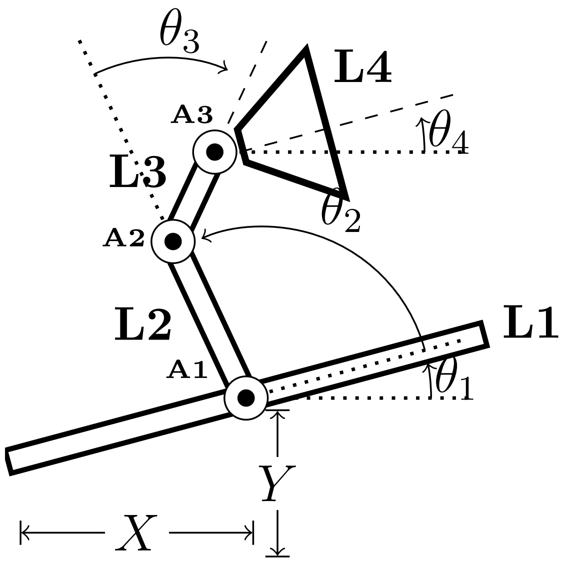


Figure 3.2: The mechanical configuration of Luxo.

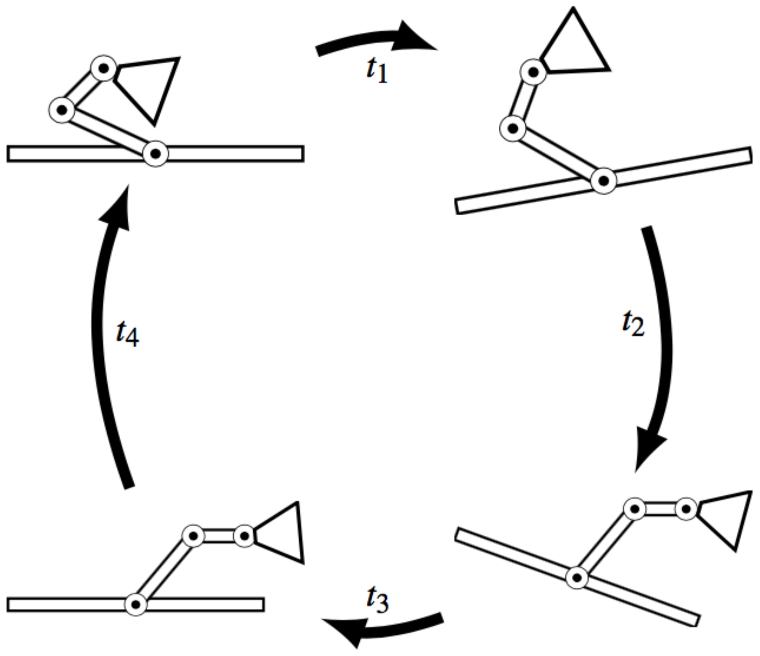


Figure 3.3: Pose control graph for jumping Luxo. The control graph is parameterized by the target poses for the 4 states and the transition durations t_1,t_2,t_3,t_4.

3.1.3 Control Scheme for Jumping

Our control scheme for Luxo, shown in Figure 3.3, is based on the periodic controller synthesis technique presented in [32]. It consists of a simple finite state machine 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.

3.1.4 Finding Motions with Simulated Annealing

Algorithm 1 gives 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.

Algorithm 1 Simulated Annealing

```
 1: function SimulatedAnnealing()
 2:     $T \leftarrow T_{max}$
 3:     $K \leftarrow K_{max}$
 4:     $s \leftarrow$ INIT()
 5:     while $K > K_{max}$ do
 6:         $s' \leftarrow$ NEIGHBOUR($s$)
 7:         $\Delta E \leftarrow \mathbf{E}(s') - \mathbf{E}(s)$
 8:         if random() < ACCEPT($T, \Delta E$) then
 9:             $s \leftarrow s'$
10:         end if
11:         $T \leftarrow$ COOLING($K$)
12:     end while
13:     return $s$
14: end function
```

The Energy Function

The quality of motions is measured by the energy function

$$\mathbf{E}(s) = -(1.0 - w_e)D - w_e H_{max}. \tag{3.1}$$

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.

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.

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:

$$\mathbf{ACCEPT}(T, \mathbf{E}(s), \mathbf{E}(s')) = \begin{cases} 1, & \text{if } \mathbf{E}(s') < \mathbf{E}(s') \\ \exp(\frac{-(\mathbf{E}(s')-\mathbf{E}(s))}{T}), & \text{otherwise} \end{cases} \tag{3.2}$$

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 (3.2), 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.

Implementation Details

We use the Bullet physics engine [1] 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.

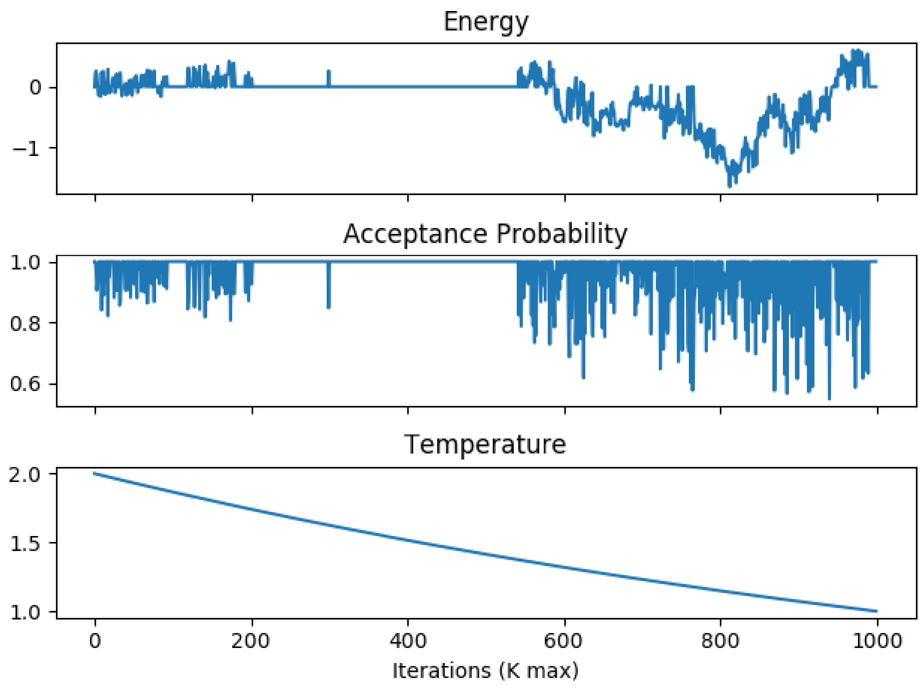


Figure 3.4: The energy, acceptance probability, and temperature values for a single run of the simulated annealing algorithm over 1000 search iterations.

3.2 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. We note that in most animation systems, a keyframe could also allows a user to define the incoming and outgoing tangents, which allows for incoming and outgoing velocities to be specified, as well as motion discontinuities, but this is not the case for our system. 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 inbetween 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 inbetween 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 inbetween frames $Y = \{Y_0, Y_1, ..., Y_m\}$, where $m = T_n - T_0 + 1$. Figure 3.5 shows a preprocessed jump sequence from our training set.

3.3 ARNN Network

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

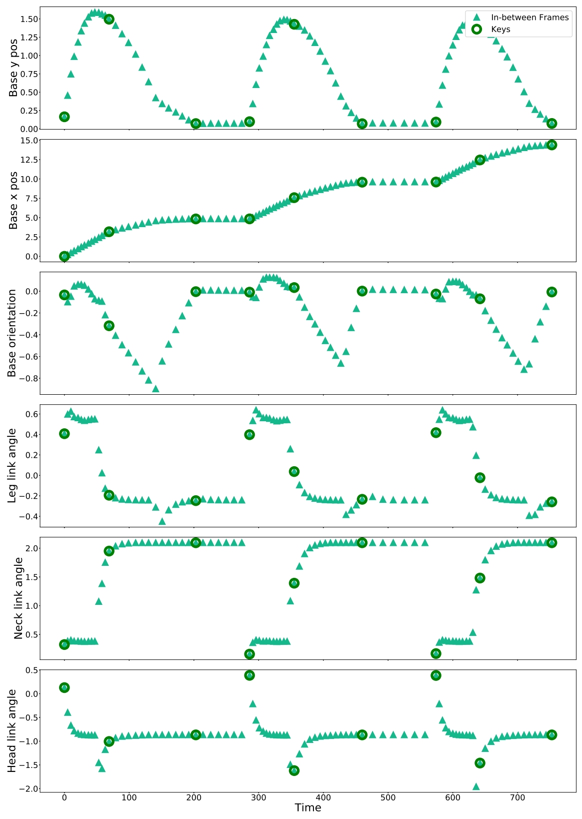


Figure 3.5: Preprocessed training data with extracted keyframes for each degree of freedom. There are 25 frames of animation for each full jump, punctuated by 6 frames of still transition between jumps. The bold open circles indicate extracted key frames.

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

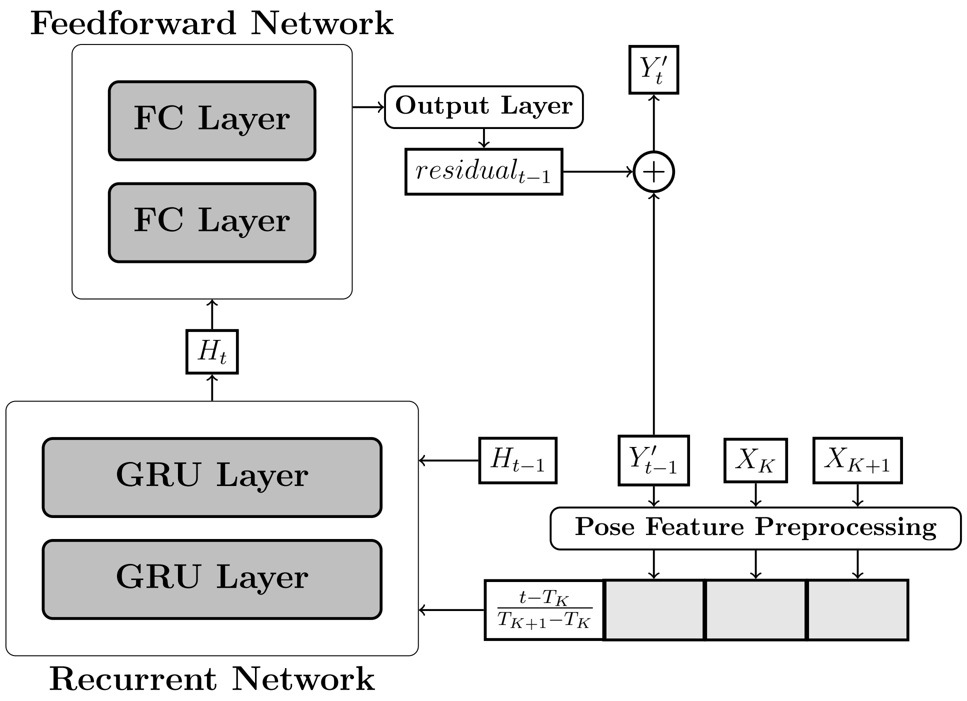


Figure 3.6: Architecture of the ARNN. The ARNN is composed of a recurrent portion and a feed-forward portion.

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 gated recurrent units (GRUS) [8]. We use scaled exponential linear unit (SELU) activations [19] between intermediate outputs. The GRU cells have 100 hidden units each, with architectures described by the following equations:

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}H_{(t-1)} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}H_{(t-1)} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t(W_{hn}H_{(t-1)} + b_{hn})) \\ H_t &= (1 - z_t)n_t + z_tH_{(t-1)} \end{aligned} \tag{3.3}$$

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 Section 4.1.

3.4 Training

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. In this section, we describe the details of our training process, including the loss function and the curriculum learning strategy we use to train the network.

3.4.1 Loss Function

To drive the network to learn to interpolate between keyframe constraints, we developed a custom loss function for the task,

$$\begin{aligned} L_{ARNN} &= 100\omega \sum_{K=1}^{n} (X_K - Y'_{T_K})^2 + MSE(Y, \hat{Y}) \\ MSE(W,Z) &= \frac{1}{N}\sum_{i=1}^{N}(W_i - Z_i)^2. \end{aligned} \tag{3.4}$$

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 inbetweening 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, inbetween 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 [5] to help the network learn better during training.

3.4.2 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 [4] 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 3.7. 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 3.7.

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 qualitative and quantitative results as shown in Table 3.1.

Training Procedure	Key Loss	Frame Loss	Total Loss
With ω annealing	0.00100392	0.00395658	0.00496051
No ω annealing	0.00120846	0.00670011	0.00790857

Table 3.1: 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.

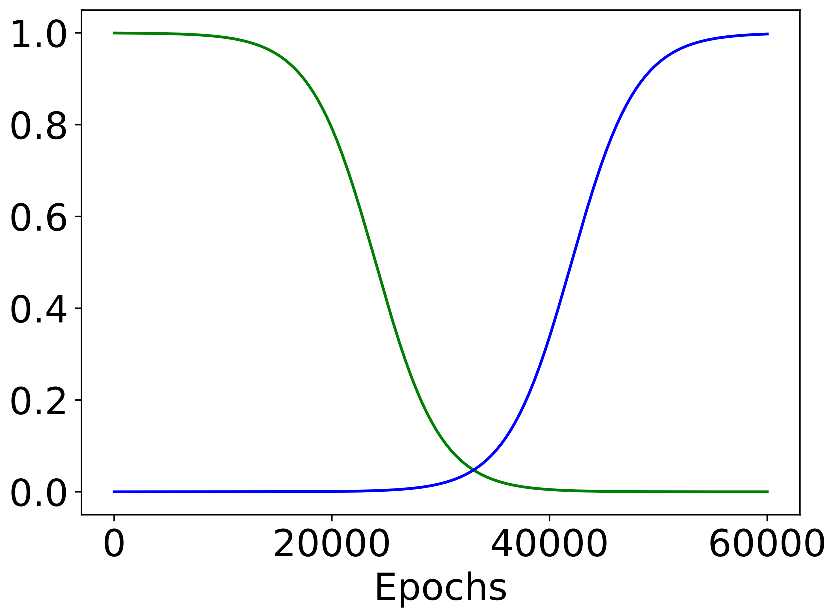


Figure 3.7: Curriculum learning schedules used to train the ARNN network during 60000 training epochs: teacher forcing ratio decay (Green Curve) and key importance weight ω annealing (Blue Curve)

3.4.3 Training Details

The final model we use 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 0-2 arbitrary keys from the sequence each time it is fed into the network. The model is optimized for 60000 epochs using Adam [18] with an initial learning rate of 0.001, 1 = 0.9, 2 = 0.999 and $\varepsilon = 10^8$ and regularized using dropout [30] 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.

Chapter 4

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 AVARS for two synthesized jumps from the test set are plotted in Figure 4.1. 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.

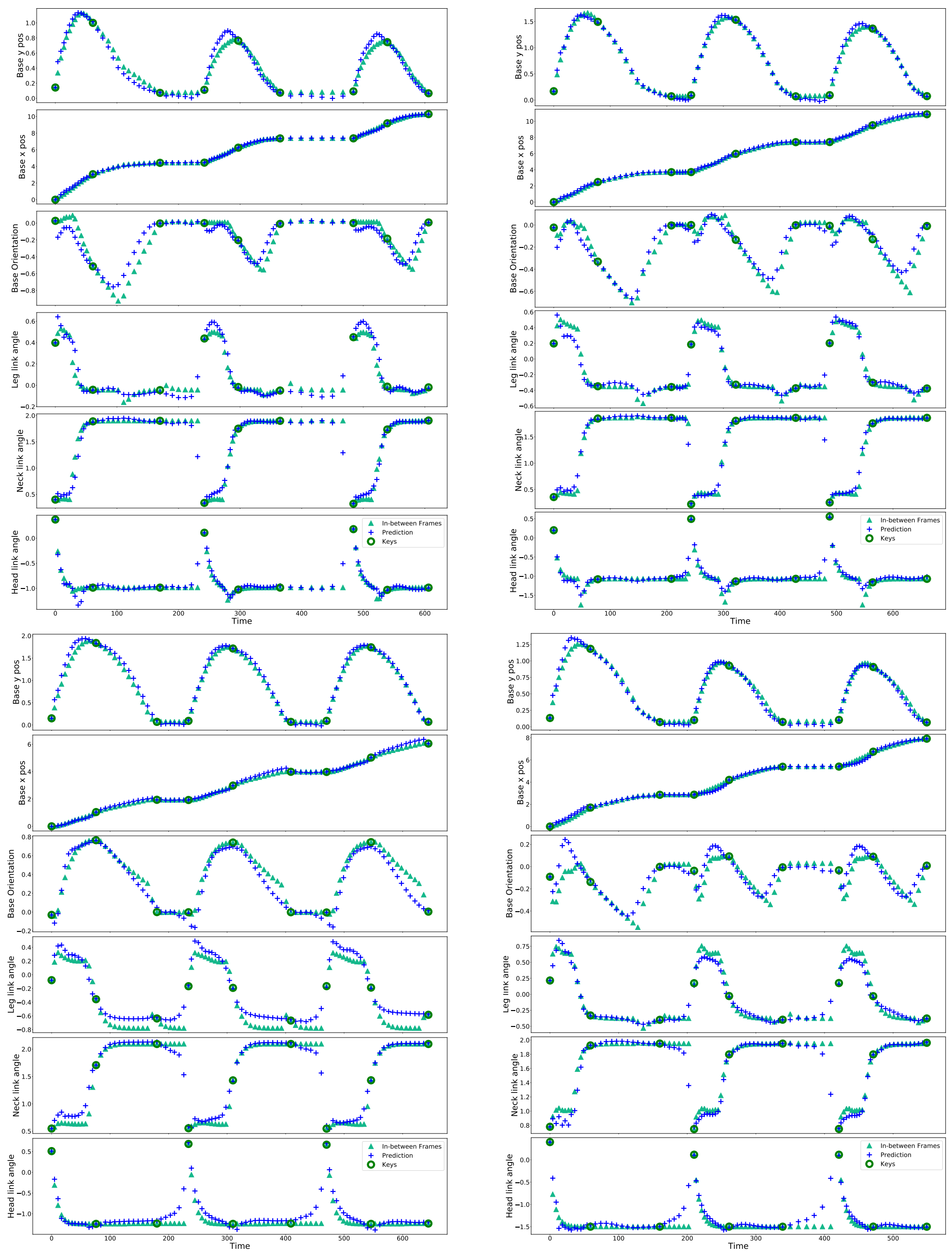
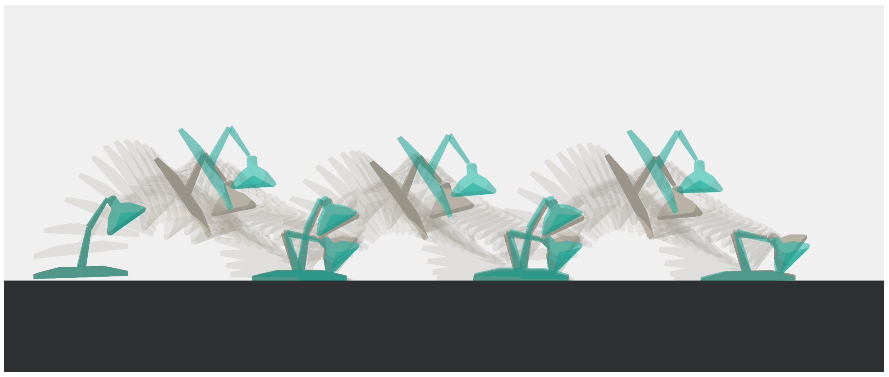
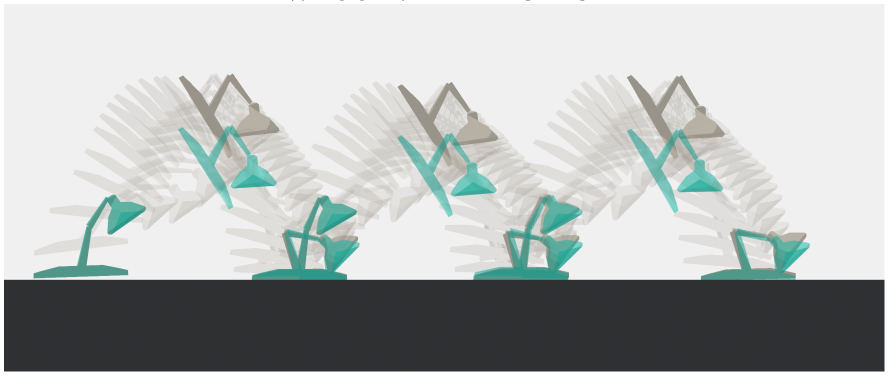


Figure 4.1: Motion reconstruction of four jumps taken from the test set. AVARS from the original simulated motion trajectory from the test set is plotted in light green. Extracted keyframes are circled in dark green. The resulting motion generated using our network is displayed in dark blue.

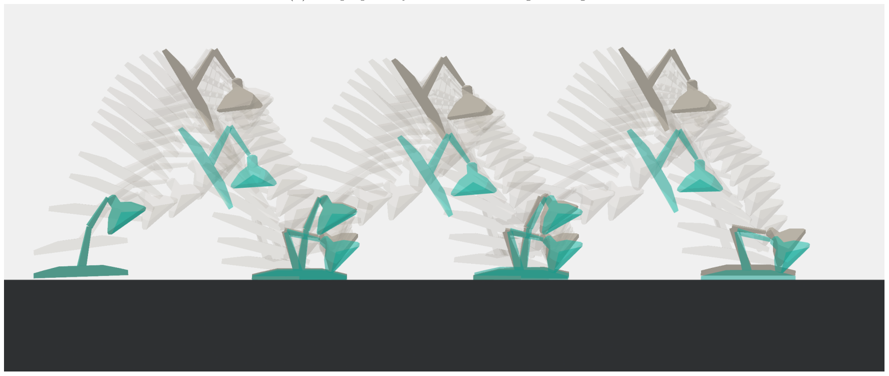
We next demonstrate generalization by applying edits of increasing amplitudes to motions in the test set. Our system produces plausible interpolated motions as we modify keyframes to demand motions that are increasingly non-physical and divergent from the training sets. Figure 4.2 shows a sequence of 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. The generated motion also appears to be natural and physically correct. The character accelerates during liftoff, decelerates nearing the apex of the jump before accelerating again at landing. Next, we modify the timing of another sequence from the test set. Timing edits are shown in Figure 4.3. Here, our system generates reasonable results. The spacing of the generated trajectory again appears to follow the laws of physics and the character accelerates and decelerates at the right moments in the jump. However, we do note that the generated motion 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

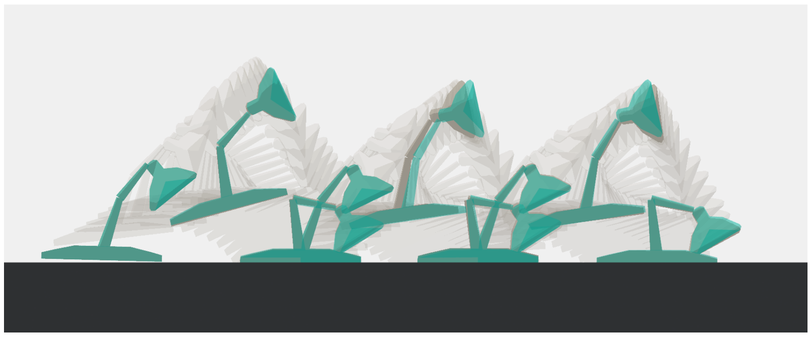


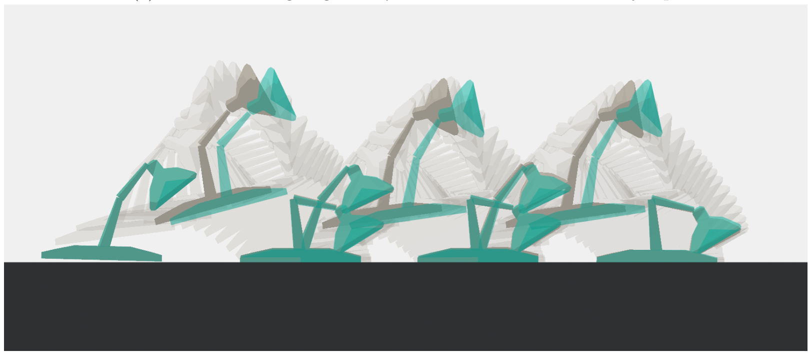
(b) Jump apex keyframed at 1.5x original height



(c) Jump apex keyframes at 1.8x original height

Figure 4.2: Height edits. Keyframes extracted from the original test jump are shown in green. The trajectory of the generated motion smoothly tracks the new apex keyframes edited to have base heights of 0.7 (top), 1.5 (middle), and 1.8 (bottom) times the original base height.


(a) Prediction using original keyframes extracted from a test jump.


(b) Prediction with 2x faster jump takeoffs.

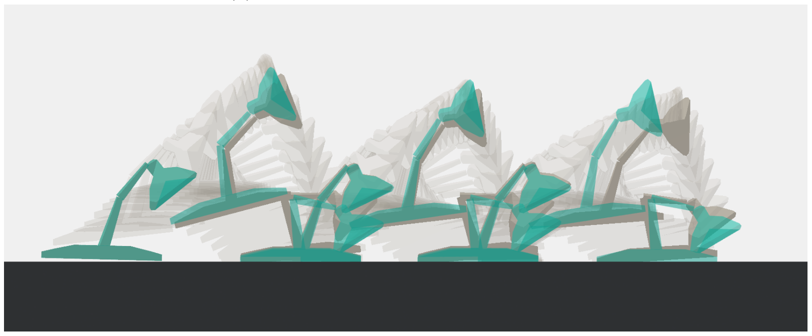

(c) Prediction with 1.5x slower jump takeoffs.

Figure 4.3: Timing edits. Input keyframes extracted from the original test jump are shown in green. The predicted pose at key locations are shown in dark gray. The top figure shows the prediction using unmodified keyframes; the keyframe of the pose at the top of the 3 jumps occur at t=13,46,79. In the middle figure, the jumps are keyed to have faster take off. The new keyframes with the same pose are newly located to be at t=7, 40, 73. The bottom figure shows the jumps with slower takeoff with the jump top keyframes shifted to be at t=19, 52, 85.

We next show that in the absence of keyframe direction, i.e, with sparse key inputs, our network is still able to output believable trajectories, demonstrating that the predictions output by the network are style-and-physics aware. In Figure 4.4, 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 is 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.

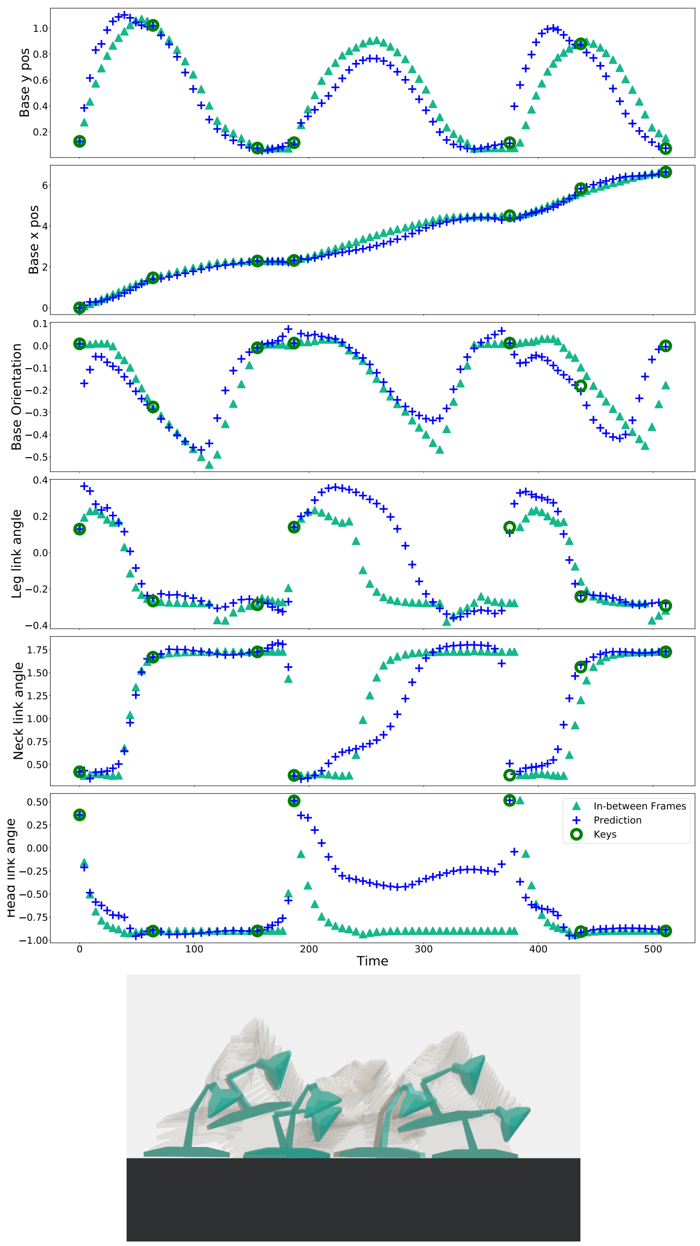


Figure 4.4: Motion generation with sparse keyframes. The apex and landing keyframes for the above jumps have been removed from the input.

The non-linearity and complexity of the motions as output by the network can also be seen in Figure 4.5. It 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.

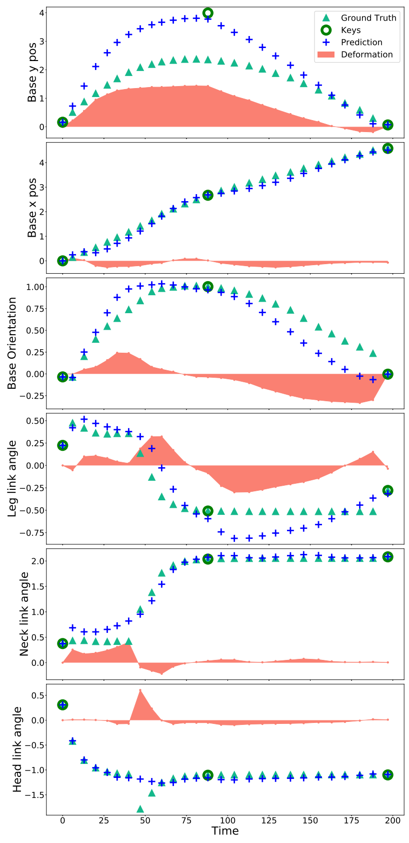


Figure 4.5: Coupled nature of the motion synthesis model. Edits to a single degree of freedom (top graph, base y position) leads to different warping functions for the other degrees of freedom.

A number of results are further demonstrated by randomly sampling, mixing, and perturbing keyframes from the test dataset, as shown in Figure 4.6.

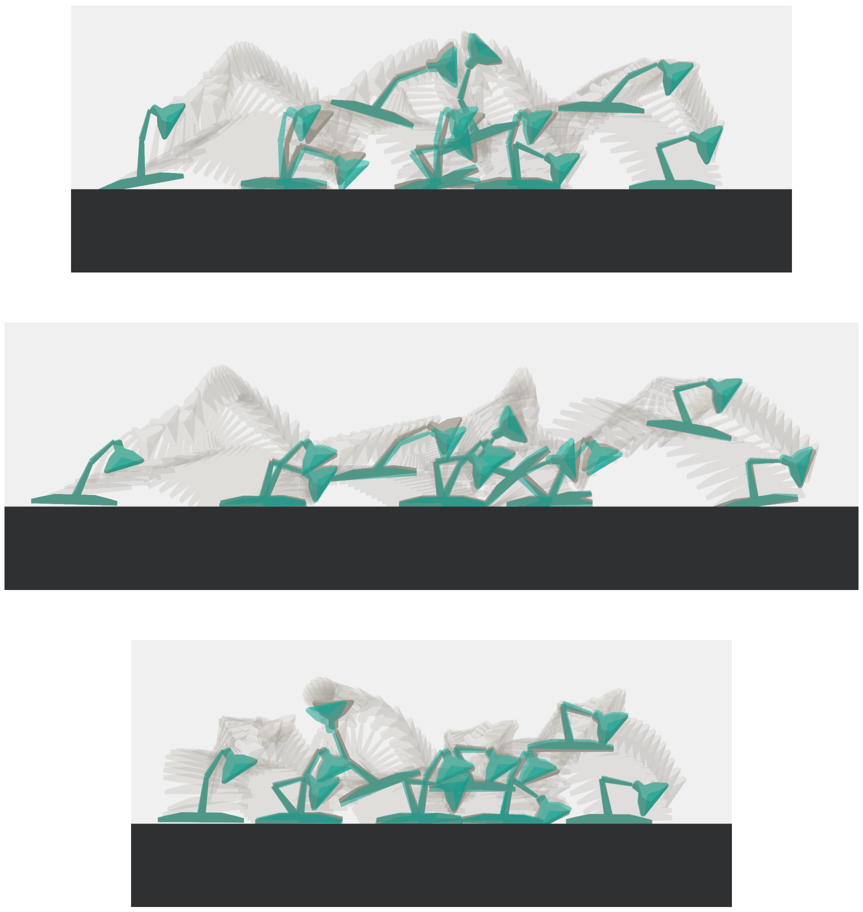


Figure 4.6: Motion synthesis from novel keyframe inputs. We created new keyframes from randomly sampled and perturbed keys taken from the test set (green). The output motion from the network is shown with predicted poses at input key locations shown in dark gray.

4.1 Comparison to Other Architectures

We evaluate 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 4.7. 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.

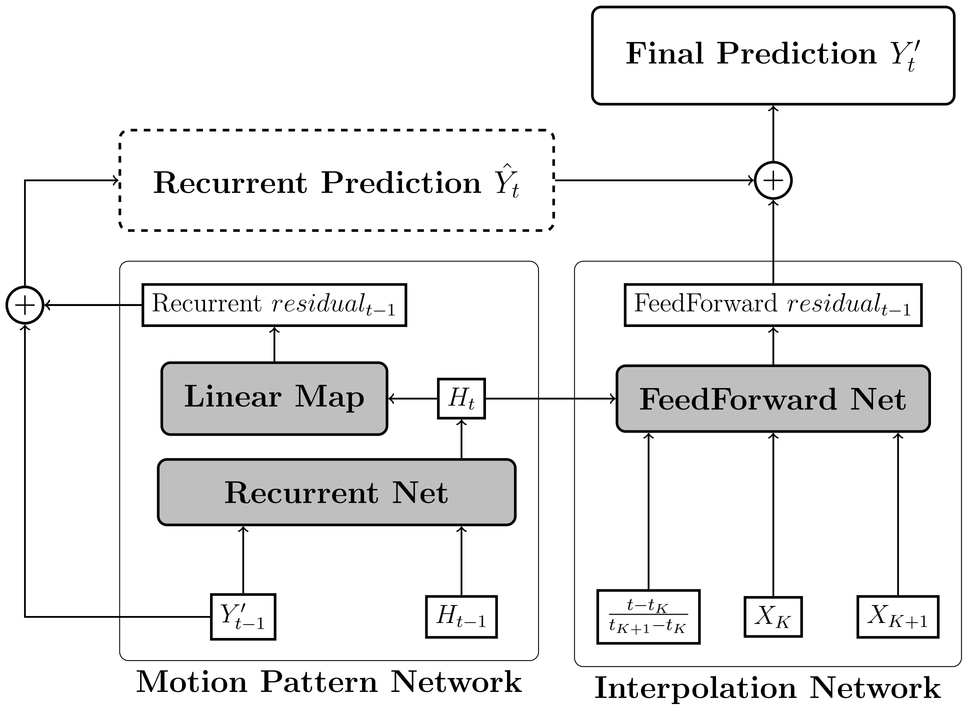


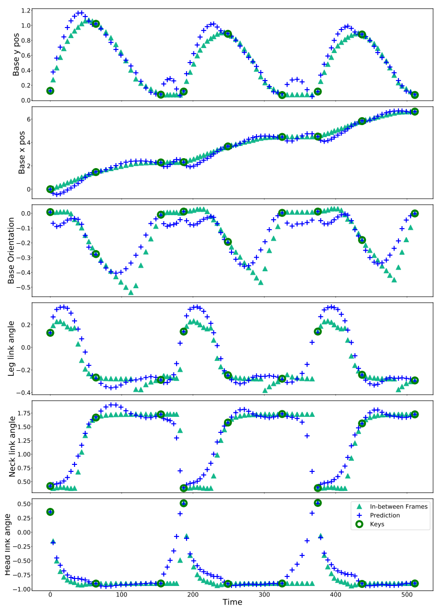
Figure 4.7: Architecture of the segregated network which combines a RNN only prediction produced by the Motion Pattern Network, with a keyframe conditioned correction produced by a feed-forward Interpolation Network.

Quantitatively, the ARNN produces the lowest total loss out of the three architectures Table 4.1 and the segregated net produces the worst loss.

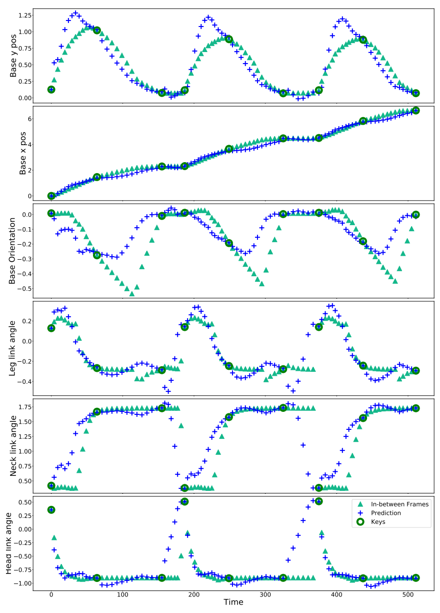
Architecture	Key Loss	Frame Loss	Total Loss
ARNN	0.00100392	0.00395658	0.00496051
No Memory Net	0.000183804	0.0081695	0.00835331
Segregated Net	0.00124264	0.0160917	0.0189122

Table 4.1: The test losses from other network architectures vs the ARNN. The ARNN produces the best overall losses.

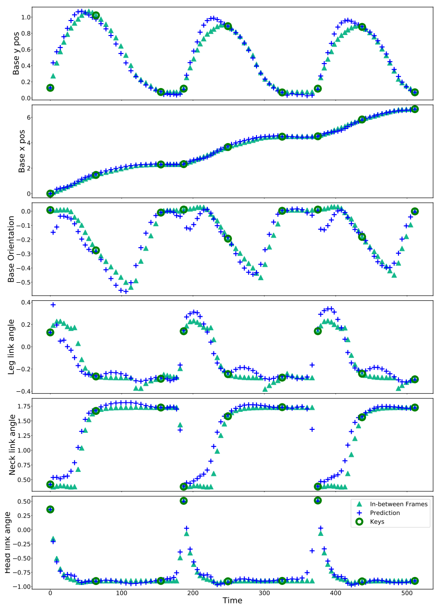
The ARNN also produces the best results qualitatively, while the feed-forward only network produces the least-desirable results due to motion discontinuities. Figure 4.8 shows the failure case when a network has no memory component. Although the feed-forward only network generates interpolating inbetweens 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 is 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.



(a)



(b)



(c)

Figure 4.8: Qualitative comparison between results for (a) a feed-forward net; (b) a segregated net; and (c) the ARNN. The ARNN and segregated nets produce smoother motions at key transitions with the use of memory.

Chapter 5

Conclusions

In this thesis, 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, e.g., [23], 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 both the physics and style 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.

The method we present, together with its evaluation, still has numerous limitations. If target keyframes go well beyond what was seen in the training set, the motion quality may suffer. We wish to further improve the motion coverage of the method via data augmentation methods, e.g., collecting physics-based motions in reduced gravity environments. While the current results represents an initial validation of our approach, we wish to apply our model to more complex motions and characters. 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 inbetweens 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.

In terms of production usability, a limitation of the current model we have developed is lack of support for partial keyframe control; the full set of AVARS for the character must be specified per keyframe for our system. However, animators working in production environments almost never specify the full set of AVARS when keyframing. This limitation is in part due to the nature of the artificial training data set we've created for this thesis, which does not include partial keyframes. In future work, we would like to to add support for partial keyframing by using more realistic animation dataset.

Lastly, there is still other significant work to be done before our system can be incorporated into production tools in terms of required training data and tackling issues that may only be observable with deployment at scale. The quality of results produced by our system is dependent on the training data we use to train the ARNN. If there is not enough training data or motion variation in the training data, the quality of the output may deteriorate. Additionally, the Luxo character we created for this thesis only has 6 degrees of freedom, but in production settings, animators often work with characters controlled by hundreds of AVARS. We hope to test the applicability of our system for production use on real animation data sets and in real production settings in future work.

Bibliography

[1] Y. Bai and E. Coumans. a python module for physics simulation in robotics, games and machine learning., 2016-2017. http://pybullet.org/. → page 15

[2] Y. Bai, D. M. Kaufman, C. K. Liu, and J. Popović. Artist-directed dynamics for 2d animation. *ACM Trans. Graph.*, 35(4):145:1–145:10, July 2016. ISSN 0730-0301. doi:10.1145/2897824.2925884. URL http://doi.acm.org/10.1145/2897824.2925884. → page 6

[3] J. Barbič, M. da Silva, and J. Popović. Deformable object animation using reduced optimal control. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 53:1–53:9, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-726-4. doi:10.1145/1576246.1531359. URL http://doi.acm.org/10.1145/1576246.1531359. → page 6

[4] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. 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, pages 1171–1179, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969239.2969370. → page 22

[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi:10.1145/1553374.1553380. URL http://doi.acm.org/10.1145/1553374.1553380. → page 21

[6] N. Burtnyk and M. Wein. Computer animation of free form images. In *ACM SIGGRAPH Computer Graphics*, volume 9, pages 78–80. ACM, 1975. → page 4

[7] S. Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the 27th Annual Conference on*

Computer Graphics and Interactive Techniques, SIGGRAPH ’00, pages 219–228, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi:10.1145/344779.344882. URL http://dx.doi.org/10.1145/344779.344882. → page 6

[8] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259. → page 20

[9] P. Coleman, J. Bibliowicz, K. Singh, and M. Gleicher. Staggered poses: A character motion representation for detail-preserving editing of pose and coordinated timing. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA ’08, pages 137–146, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association. ISBN 978-3-905674-10-1. URL http://dl.acm.org/citation.cfm?id=1632592.1632612. → page 5

[10] L. Crnkovic-Friis and L. Crnkovic-Friis. Generative choreography using deep learning. *CoRR*, abs/1605.06921, 2016. URL http://arxiv.org/abs/1605.06921. → page 7

[11] B. Dalstein, R. Ronfard, and M. van de Panne. Vector graphics animation with time-varying topology. *ACM Trans. Graph.*, 34(4):145:1–145:12, July 2015. ISSN 0730-0301. doi:10.1145/2766913. URL http://doi.acm.org/10.1145/2766913. → pages ix, 4, 5

[12] K. Fragkiadaki, S. Levine, and J. Malik. Recurrent network models for kinematic tracking. *CoRR*, abs/1508.00271, 2015. URL http://arxiv.org/abs/1508.00271. → page 7

[13] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović. Style-based inverse kinematics. *ACM Trans. Graph.*, 23(3):522–531, Aug. 2004. ISSN 0730-0301. doi:10.1145/1015706.1015755. URL http://doi.acm.org/10.1145/1015706.1015755. → page 7

[14] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20. ACM, 1998. → page 8

[15] K. Hildebrandt, C. Schulz, C. von Tycowicz, and K. Polthier. Interactive spacetime control of deformable objects. *ACM Trans. Graph.*, 31(4):

71:1–71:8, July 2012. ISSN 0730-0301. doi:10.1145/2185520.2185567. URL http://doi.acm.org/10.1145/2185520.2185567. → page 6

[16] D. Holden, J. Saito, and T. Komura. A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph.*, 35(4): 138:1–138:11, July 2016. ISSN 0730-0301. doi:10.1145/2897824.2925975. URL http://doi.acm.org/10.1145/2897824.2925975. → pages ix, 7, 8, 9

[17] D. Holden, T. Komura, and J. Saito. Phase-functioned neural networks for character control. *ACM Trans. Graph.*, 36(4):42:1–42:13, July 2017. ISSN 0730-0301. doi:10.1145/3072959.3073663. URL http://doi.acm.org/10.1145/3072959.3073663. → pages 7, 8

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980. → page 23

[19] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017. URL http://arxiv.org/abs/1706.02515. → page 20

[20] A. Kort. Computer aided inbetweening. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pages 125–132, New York, NY, USA, 2002. ACM. ISBN 1-58113-494-0. doi:10.1145/508530.508552. URL http://doi.acm.org/10.1145/508530.508552. → page 4

[21] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, Aug. 2004. ISSN 0730-0301. doi:10.1145/1015706.1015760. URL http://doi.acm.org/10.1145/1015706.1015760. → page 7

[22] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, July 2002. ISSN 0730-0301. doi:10.1145/566654.566605. URL http://doi.acm.org/10.1145/566654.566605. → page 6

[23] Z. Li, Y. Zhou, S. Xiao, C. He, and H. Li. Auto-conditioned lstm network for extended complex human motion synthesis. *arXiv preprint arXiv:1707.05363*, 2017. → pages 8, 36

[24] J. Min, Y.-L. Chen, and J. Chai. Interactive generation of human animation with deformable motion models. *ACM Trans. Graph.*, 29(1):9:1–9:12, Dec. 2009. ISSN 0730-0301. doi:10.1145/1640443.1640452. URL http://doi.acm.org/10.1145/1640443.1640452. → page 7

[25] J.-C. Nebel. Keyframe interpolation with self-collision avoidance. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation and Simulation '99*, pages 77–86, Vienna, 1999. Springer Vienna. ISBN 978-3-7091-6423-5. → page 5

[26] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017. → page 8

[27] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5): 32–40, Sept. 1998. ISSN 0272-1716. doi:10.1109/38.708559. URL http://dx.doi.org/10.1109/38.708559. → page 7

[28] A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (ToG)*, 23(3):514–521, 2004. → page 7

[29] C. Shen, T. Hahn, B. Parker, and S. Shen. Animation recipes: Turning an animator's trick into an automatic animation system. In *ACM SIGGRAPH 2015 Talks*, SIGGRAPH '15, pages 29:1–29:1, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3636-9. doi:10.1145/2775280.2792531. URL http://doi.acm.org/10.1145/2775280.2792531. → page 5

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313. → page 23

[31] C. D. Twigg and D. L. James. Many-worlds browsing for control of multibody dynamics. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. doi:10.1145/1275808.1276395. URL http://doi.acm.org/10.1145/1275808.1276395. → page 6

[32] M. van de Panne, R. Kim, and E. Flume. Virtual wind-up toys for animation. In *Proceedings of Graphics Interface '94*, pages 208–215, 1994. → page 13

[33] X. Wei, J. Min, and J. Chai. Physically valid statistical models for human motion generation. *ACM Transactions on Graphics (TOG)*, 30(3):19, 2011. → page 7

[34] B. Whited, G. Noris, M. Simmons, R. Sumner, M. Gross, and J. Rossignac. Betweenit: An interactive tool for tight inbetweening. *Comput. Graphics Forum (Proc. Eurographics)*, 29(2):605–614, 2010. → page 4

[35] Wikipedia. Avar (animation variable) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Avar%20(animation%20variable)&oldid=815219789, 2018. [Online; accessed 18-April-2018]. → page 1

[36] A. Witkin and M. Kass. Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, pages 159–168, New York, NY, USA, 1988. ACM. ISBN 0-89791-275-6. doi:10.1145/54852.378507. URL http://doi.acm.org/10.1145/54852.378507. → page 6

[37] A. Witkin and Z. Popovic. Motion warping. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 105–108, New York, NY, USA, 1995. ACM. ISBN 0-89791-701-4. doi:10.1145/218380.218422. URL http://doi.acm.org/10.1145/218380.218422. → page 5

[38] A. Witkin and Z. Popovic. Motion warping. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 105–108. ACM, 1995. → page 7