Data-Driven Gaze Animation using Recurrent Neural Networks

A. Klein
Supervisors: Z. Yumak[1], A.F. van der Stappen[1], A. Beij[2]

[1]Utrecht University, Utrecht, The Netherlands
[2]Guerrilla Games, Amsterdam, The Netherlands

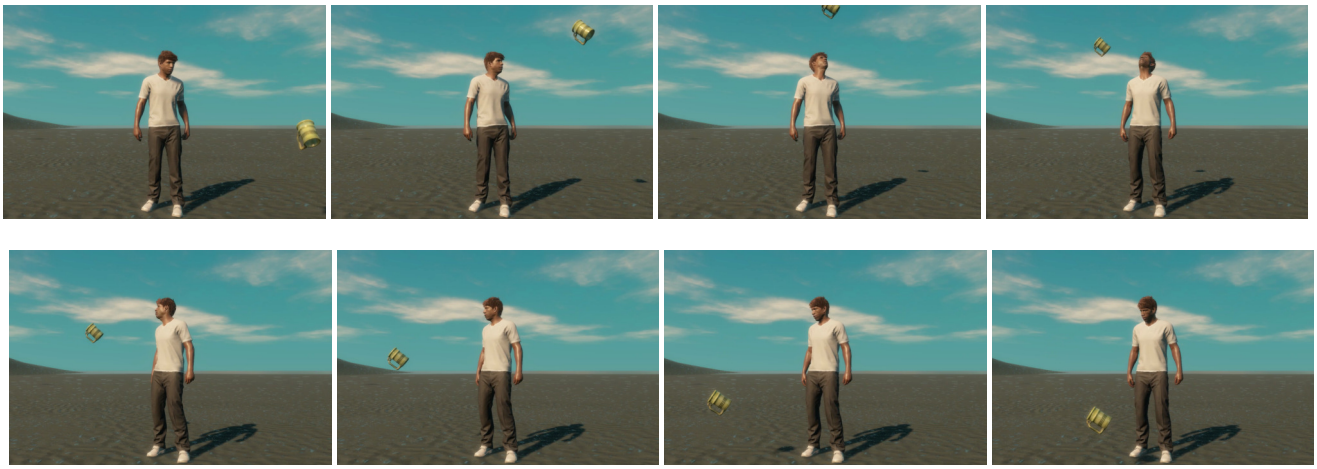

Figure 1: Smooth motion is learned when the target moves overhead.

Abstract

We present a real-time gaze animation system using recurrent neural networks. Both motion capture data and video from a head-mounted camera are used to train the network to predict the motion of the body and the eyes. The system is trained separately on different poses, e.g. standing, sitting, and laying down, and is able to learn constraints on movement per pose. A simplified version of the neural network is presented, for scenarios which allow for lower detail gaze animation. We compare various neural network architectures and show that our method has the capability to learn realistic gaze motion from the data, while maintaining performance. Results from a user study conducted among game industry professionals, shows that our method significantly improves perceived naturalness of the gaze animation, compared to a manually created procedural gaze system.

CCS Concepts

• ***Computing methodologies*** → *Motion Capture;*

1. Introduction

Gaze animations are regarded as a solved problem in the video game industry, because manual designed gaze animations were sufficient for virtual characters for the past decades. However, creating robust gaze animations is a time consuming task for animators, because such a system needs to work in many different scenarios, leading to complex and messy animation systems that are hard to maintain. Additionally, on closer inspection these animations are not as close to realistic behaviour as hoped.

Data-driven animation and machine learning have shown promising results in improving animation quality, without creating an explosion in complexity to the animation system, from which procedural animation systems often suffer. Recently, many promising results have been presented by applying neural networks in computer graphics. For example, neural networks have been used for pose classification and to animate full body locomotion in complex environments, based on a large data set of motion capture data [FLFM15, HKS17, PBYVDP17, MBR17, ZSKS18, PGA18, PALvdP18]. Therefore, the main challenge of this thesis is how to create data-driven gaze animations.

This poses the following questions:

- How can we teach a computer look-at behaviours?
- What are the requirements on the data to learn look-at behaviours?
- Can this approach be fast enough to be used in real-time gaming environments?

This thesis proposes a machine learning approach for gaze animation. We train a neural network on multi-modal data from motion capture and eye tracking from a head-mounted camera. The neural network predicts the next frame of the gaze animation from a given look-at target and the current pose of the character. Special scenarios were recorded to obtain the right data for such a gaze model.

2. Related Work

In this section we review previous work that is relevant for our research, starting with the gaze behaviour in humans that we intend to recreate. Next, an overview of techniques for gaze animation is given. Finally, we review methods that have used neural networks for animation.

2.1. Gaze Behaviour

There is a large body of research on gaze behaviour, both physiological and psychological [LZ15]. A key concept here is *gaze*, which is the direction of the eyes and head to a specific target. *Gaze shifts* are the movements of the eyes and head, and potentially shoulders, spine and hips, to redirect the gaze to another target.

Saccades are rapid shifts in the eye rotation, making them the most noticeable eye movements. These movements are performed very similarly among people. Saccades are very rapid movements, where even large movements are performed in a tenth of a second. Large saccades of 30° are quite rare, while 5° to 10° is common, taking 30 to 40 ms. Saccade latency, i.e. the time it takes to start moving them, is between 100 and 200 ms. The time between saccades is normally around 150 ms. Most other properties of saccades, including undershooting the target, and small drifts after the saccade, are often not noticeable in common interactions between people [RAB*14].

The *Vestibulo Occular Reflex* (*VOR*) is used to stabilise the eyes during head motion. It is very quick, since it is controlled by the nearby inner-ear vestibular neurons. Eyes can be rotated faster than the head, which makes both movements seem to occur almost simultaneously. VOR should rotate the eyes to exactly counter the head movement, although head roll movement (rotating along the direction of your nose) can be ignored for efficiency [RAB*14].

Smooth pursuit stabilises the eyes to track a moving object, like reading the text on the side of a moving bus [RAB*14]. Smooth pursuit is used to centre an image on the retina. It has a latency of 80 to 130 ms and breaks down at high velocities. Because smooth pursuit is only used in quite specific situations (like reading from a moving surface), it is not often noticed.

Normally, both eyes look in the same direction. The exception is *vergence*. Vergence happens when the eyes need to focus on a nearby object and the eyes need to rotate towards each other to keep the vision sharp.

Eyelid movement mostly comes in the form of blinks. There are roughly three kind of blinks: spontaneous, voluntary, and reflexive. Spontaneous blinks are based on cognitive and social activities, and can vary wildly in timing. Voluntary blinks occur from time to time to keep the eyes from drying out. Reflexive blinks are a protective mechanism, when something touches the eye or gets very close to the eye. Blinking behaviour can be modelled by a Poisson set, but blinks often co-occur with eye and head movement. Eyelid displacement especially accompanies vertical saccades, i.e. looking up or down [RAB*14].

For larger gaze movements, both the eyes and head are involved. Normally, eyes move first after 200 ms latency, followed by the head 20 to 50 ms later. However, when looking towards a predictable target, the head turns first and the eyes move afterward [LZ15]. If a person knows beforehand that he will look at a certain target, this is called a predictable target, e.g. when a person changes his gaze to another person in a group conversation. An unpredictable target would mean that the target was not known in advance and the gaze should move to this target quickly. An example of an unpredictable target is a car honking in traffic. How eye movement and head movement are combined differs between people [LZ15].

Gaze is often used to communicate the internal state of a person, like showing interest and one's emotional state. During conversation it can help to indicate turn-taking . In turn, emotions can have an effect on gaze behaviour [LMK04, LM10]. It has been shown that the meaning of gaze behaviour can change, depending on cultural differences [RAB*14]. Not following the cultural norm in mutual gaze can affect engagement in conversation. During conversation gaze, nods, head movements, blinks, and eyebrow gestures can be used to provide context to speech.

Visual attention is the way people select their gaze targets. It selects the locations to look at, based on interest and importance [RAB*14]. Visual attention, together with visual perception, controls of the visual sensory system to observe the world. People tend to move their attention to objects that are large, bright, and moving in the peripheral vision [PVZ08].

2.2. Gaze Animation

Humans use a combination of eye and head movement, and optionally move their shoulders and hips, to look at a target. In animation, this kind of movement is often explicitly modelled by an animator. The animator sets the rules which govern the animation. These manually made gaze animations are often good enough for the animator's purpose, but they are labour-intensive and hard to maintain. Research has focussed on limiting the amount of manual labour that is required to make these systems. For a more extensive overview of look-at animation, see the overview written by Ruhland et al. [RAB*14].

To add natural head movement to speaking agents, Brkic et al. [BSPP08] developed a system that would add nods and head swings to the facial gestures that were already in the system. The

facial gestures were based on the English text the agent had to say. The nods and swing were modelled with sinus functions, based on analysis of video footage. Although the added head movement was perceived as more realistic, it is a rather simple approximation that could be improved upon.

Eye movement for gaze, especially when combined with head movement for gaze, can be more challenging when animating stylised characters. Assumptions about eye movement for realistic characters break when applied to these stylised characters. Various artifacts can occur when the size, location, and orientation of the eyes is changed relative to the head, e.g. cross-eyedness, eyes that move too fast, stuck eyes, and eye divergence. Pesja et al. [PMG13] improved the gaze animation to solve these artifacts. They also provided various controls for an animator to adjust the gaze behaviour.

Later, Pesja et al. [PRMG16] developed a system where gaze targets could be inferred from motion capture data, where such information was not yet present. These inferred gazes could also be edited by an animator to change the apparent intention of a character. By detecting maxima in the head velocity, it is possible to determine when the gazing events take place. The look-at target is then selected, based on a combination of the movement, action, and environment of the character. This is then used as input data, which could later be edited by an animator. However, the gazes are only heuristically inferred from motion capture data, instead of based on captured data.

The selection of the look-at target is an active field of research in computer animation. Kuhllar and Badler [KB01] developed a system based on psychological research. It takes into account the field of view of the virtual character and different gazing behaviours for different intentions. Targets are selected based on the task of the virtual character, as well as the objects surrounding the character. When no tasks are at hand, spontaneous looking is the default behaviour and the target alternates between the surrounding objects. Cafaro et al. [CGV09] captured and analysed videos of public places to see where people look to develop a base line gaze behaviour. They created a model that recreated the observations from their video data, which animated idle gazing. Both works on look-at targets do not consider the look-at animation and assume that such a system is already in place.

2.3. Data-Driven Animation

Deep learning for animation has seen a great increase of interest in the past years. These approaches try to lower the amount of manual editing by an animator and improve realism by learning from recorded humans. Below, this research has been split in three categories: learning facial animation, learning locomotion using reinforcement learning, and deep neural networks to learn locomotion from recorded data.

Some gaze systems provide head and eye movement, given speech audio as input. These models are trained on a combination of speech and video/motion capture data, to learn to predict the gaze behaviour based on the speech input. Lee et al. [LBB02] created a statistical model from eye tracking data. This model could enhance existing animations with eye movement for speaking and listening agents. This was one of the first approaches to make use of recorded data to create a model for gaze animation. Their focus is on eye cues during social interaction, where they make a distinction in eye behaviour between talking or listening. Eye movement was recorded and a statistical model was fitted to the data. This model was then used to determine saccade velocities and gaze durations. However, this model is limited to gaze behaviour during social interactions.

Le et al. [LMD12] developed a hybrid speech-driven system for head and eye movement. Three separate systems were used for head motion, eye gaze, and eyelid motion. A Gaussian mixture model is used for head motion, Nonlinear Dynamic Canonical Correlation Analysis for eye gaze, and a linear regression model for the eyelids. The models were trained on a combination of audio, motion capture, and video data. Although these systems are interconnected, they do not offer a unified system for gaze, limiting the resulting behaviour of the system.

Recently, deep learning was used to predict facial animation from audio input. Taylor et al. [TKY*17] use a recurrent neural network with a sliding window, which trains on phonemes extracted from the audio input. The model then learns from video to control the face of a character based on the phonemes. They showed that a deep neural network can be trained to perform facial animation. The network even generalises enough to be applicable to more stylised characters and extrapolate to handle speech of speakers not present in the training data. Their current system cannot handle different emotional states for the character out of the box, but the output can be easily edited to add more emotion to the animation.

Laine et al. [LKA*17] used deep convolutional neural networks (CNN) to create complete virtual 3D facial performances from video alone. An actor needs to play out a set of facial motions, from which the CNN can learn. This data is augmented by distorting it, to make the network resistant to variations within the input data. After training the model, it could recreate a test video convincingly, which shows the power of CNNs to generalise on a limited set of input data. The authors note that their selected loss function might not necessarily give low scores to actual natural-looking facial animation. Also, the neural network can only capture the facial features of one person and does not generalise well to different faces.

DeepLoco [PBYVDP17] uses Deep Reinforcement Learning to train a bipedal character to walk a path, dribble a ball, and avoid moving obstacles. They train two controllers: one for low-level control and one for high level control. The low-level controller learns to move the feet, while the high level controller focusses on learning path finding or ball dribbling. They show that this hierarchical model is capable to learn both the low-level locomotion and high-level tasks. The trained model is even resilient to minor perturbations. The hierarchical controllers make their model more modular and easier to train than one big, deep model. To provide a phase to the network, a specialied phase selection network has been added to the model. This was later expanded in [PALvdP18] to learn various actions, like kicking, throwing a ball, and back flips. They included motion capture data in the learning pipeline, to guide the learning process. They also show the adaptiveness to perturbations after training.

Deep learning has also been employed to recognise body poses in videos and motion capture data, as well as to predict the

next pose in a video. The model used by Fragkiadaki et al. [FLFM15] uses a recurrent neural network (RNN), based on the Long Short-Term Memory (LSTM) architecture, and is augmented with encoder-decoder functionality to ensure that the model does not underfit, coined Encoder-Recurrent-Decoder (ERD). Recognition of body poses in motion capture data has temporal dependencies, which are captured by the LSTM. Martinez et al. [MBR17] expanded on the idea of RNNs for animation and added residual connections (direct connections between the input and output) to the network. Adding the residual connections changes the learning problem from predicting the next frame to predicting the change between the current frame and the next. They show that this change in the RNN's architecture removes the need for the encoding and decoding layers used in the ERD.

In the past few years, there has been an increasing interest in deep learning full body locomotion. Some state-of-the-art models are described below. A common problem in learning locomotion is the *phase* of the motion, i.e. the fact that one leg is moving at a time. If no care is taken, a model cannot recognise which foot to use, resulting in unnatural foot-sliding, i.e. using both feet simultaneously. Holden et al. [HSK16] created a deep learning framework and showed how to train a humanoid character to perform various tasks, including walking, kicking, and punching. They used data from various motion capture databases. They show how to apply motion editing using their deep networks, so that can they combine a regular walking network with a particular walking style of another network, or add various constraints to a trained model. Disambiguation of the locomotion is provided by adding the timing of foot contact with the floor to the training data.

Recently, Holden et al. [HKS17] used motion capture data to train a phase-functioned neural network (PFNN) for walking animations of a user-controlled character. Motion capture data was annotated with terrain height data. The user-input was reconstructed from the motion capture data and added as input to the training data. A phase function was added to the neural network, so the network could train separately for each phase. Without the added phase, the network could not learn correctly when to use the left foot and when to use the right foot, resulting in foot-sliding. However, the phase function makes training significantly harder. The PFNN can be evaluated fast enough for interactive control by a user. To reach this performance, high-level details were lost in the network's representation. Like many machine learning models, the PFNN has trouble extrapolating beyond the presented training data, for example in situations which are physically impossible. The PFNN was later extended to work for quadrupeds [ZSKS18], using a generalised version of PFNN, called the Mode-Adaptive Neural Network (MANN).

3. Background

In the past, artificial neural networks have been used for animation generation. In this section, we give an overview on relevant artificial neural network architectures. We start with an introduction to neural networks and extend that to deep neural networks. Then, recurrent neural networks are explained.

3.1. Deep Neural Networks

Artificial neural networks are powerful machine learning tools, that have become popular over the past years. The technique is inspired by biological neurons, as in the brain. They are capable of learning a function from examples, given a set of inputs and respective outputs of that function. An example can be presented at the input neurons, which are connected to the neurons in the next layer, which will give off a real valued signal. Each connection between neurons i and j has a weight w_{ij}, which get multiplied with the signal from neuron i. The receiving neuron also has a bias b_i, which is added to the incoming signal. An activation function then determines the final output value of a neuron, given its combined input signal. Moving the signal through the layers is called *Forward Propagation* [GBC16].

In a formal way, we can see an input example as a vector x. The weights between two layers of neurons is matrix W and vector b are the biases of a layer. The output of a layer l can thus be defined as:

$$h_l = \sigma(W_l x + b_l)$$

where σ is a non-linear activation function.

A neural network with only two layers, an input and output layer, can only approximate linear functions. With one layer between the input and output layer, called the hidden layer, a neural network can approximate any function, as proven in the universal approximation theorem [GBC16]. Such networks are often called *multilayer perceptron (MLP)*. When there is more than one hidden layer, the network is called a deep neural network. Then, forward propagation for n layers would be:

$$h_l = \sigma(W_{n-1}...\sigma(W_1\sigma(W_0 x + b_0) + b_1)... + b_{n-1})$$

Neural networks with one hidden layer have no theoretical limit on the functions they can approximate. However, given enough capacity, deep neural networks can often obtain better learning results with less training time and with fewer neurons overall. In deep neural networks, we can also easily see why a non-linear activation function is needed. Without it, we could simply multiply and add the weights and biases together to one matrix again, which is itself just linear.

Many algorithms have been proposed to train neural networks. Often, some form of gradient descent is used to gradually improve the results of the network. First, an input is presented and forward propagation is used to generate the output. This output is then compared to the expected value. From this, a loss value is computed, for example the squared error:

$$\varepsilon_t = (h_t - y_t)^2$$

where h_t is the output of the neural network and y_t the expected value for time step t. From this loss, a gradient is computed using *backpropagation*, and a gradient descent algorithm is used to change the weights and biases in the network, in order to decrease the error in the output.

Like many other machine learning techniques, neural networks have the tendency to overfit on the data. When overfitting occurs, the model can generate the correct output for the training data, but has failed to learn the general rule behind the data. Overfitting is

caused by training models that are too complex for their task and can use their complexity to memorise the training data, rather than learn the rule. A plethora of methods has been proposed to help neural networks generalise better. First, various activation functions have been proposed and compared, e.g. *Rectified Linear Units (ReLU)* [GBC16], defined as:

$$ReLU(x) = \max(0, x)$$

Second, there is the choice of the optimisation algorithms. A popular algorithm for deep neural networks is Stochastic Gradient Descent (SGD) [GBC16]. Later, other algorithms have been proposed to improve convergence of deep networks, like AdaGrad, AdaDelta [GBC16], and ADAM [KB14].

Another method to improve generalisation is by regularisation, which often limits the capacity of the model, to prevent overfitting. At first overfitting can be limited by a lower number of neurons and layers for the task at hand, so the neural network will not be overcomplex and be able to just memorise all the training data, without any need for generalisation. However, if the model is too simple, it does not have the capacity to capture the rule behind the data, which is called *underfitting*. If a model can overfit the data, regularisation can then improve generalisation.

A recent popular method is *Dropout* [SHK*14], which randomly turns off neurons in the network during training. This ensures that neurons within a layer cannot co-adapt and become dependent on each other, which causes them to specialise in one task. Another form of regularisation is to add noise to the training data, so the neural network cannot fixate on the training data too much [GBC16].

When using a validation set to check the accuracy of a neural network, we can also use *Early Stopping* [GBC16]. While training the model, the error on both the training and validation set goes down. However, at some point the model starts to overfit, still minimising the error on the training set, but increasing the error on the validation set. When we detect this moment, we can stop training, as we do not expect to find a model after overfitting sets in.

3.2. Recurrent Neural Networks

Recurrent neural networks (RNNs) are speciallised network architectures for sequential data [GBC16, LBH15], e.g. data over time. RNNs are MLPs with connections going back to the same or previous layers. Therefore, RNNs do not only consider the current input, but also past inputs. If the recurrent connections would be unfolded, the network is an MLP taking inputs from multiple time steps. Thus, parameters used in one timestep can also be used for the next timestep and that this will generalise better on sequences of different lengths. Depending on the architecture of the network, output can be provided at every timestep or only at the end of a sequence. To train an RNN, the network has to be unfolded for some number of timesteps in order to propagate the error back through time (see Figure 2). If the network is trained on too few timesteps, it might not be able to learn dependencies over time.

An RNN is by definition a deep neural network, due to the recurrent connections. Yet, when an RNN has more than one hidden layer, it is called a *deep recurrent neural network* [PGCB13]. Similar to the difference between MLPs with one hidden layers and deep MLPs, deep RNNs have more than one hidden layer. This can help the network extract higher level features from the input, improving the predictive abilities of the network.

3.2.1. Long Short Term Memory

A major problem in deep learning and specifically in RNNs, is that gradients can vanish or explode during training, since they can be multiplied many times with the same weights during training. This has been a difficult problem to overcome and made it impossible to learning long-term dependencies. To solve this problem, a special kind of memory cell was proposed by [HS97], called *Long Short-Term Memory (LSTM)*. These memory cells can hold their own state over time and control how much the state is affected by the input, how much of the state should be forgotten, and how much of the cell content is passed as output. LSTM is able retain the gradient over a longer period of time, because it can control the impact new inputs have on its memory.

For a more formal description of the LSTM, we will follow the description given by [CGCB14]. The LSTM has three gates, called the *input gate*, the *forget gate*, and the *output gate*. The output of one LSTM cell j at time t is given by:

$$h_t^j = o_t^j \tanh(c_t^j)$$

where o_t^j is the output gate and c_t^j the updated memory. The output gate is defined as;

$$o_t^j = \sigma(W_o x_t + U_o h_{t-1} + b_o)^j$$

where σ is a logistic sigmoid function, W_o and U_o are weight matrices, b_o the output biases. x_t is the input at time t and h_{t-1} the output from the previous time step.

To update the memory cell c_t^j, the cell uses the forget gate to remove data from the memory and the input gate to control how much the new memory (based on the input) will affect the current memory:

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j$$

where f_t^j and i_t^j are the forget gate and input gate, respectively, defined as:

$$f_t^j = \sigma(W_f x_t + U_f h_{t-1} + b_f)^j$$
$$i_t^j = \sigma(W_i x_t + U_i h_{t-1} + b_i)^j$$

and the new memory is defined as:

$$\tilde{c}_t^j = \tanh(W_c x_t + U_c h_{t-1})^j$$

This comes to a total of eight weight matrices, four bias vectors, and one cell state vector.

3.2.2. Gated Recurrent Units

Gated Recurrent Units (GRUs) [CGCB14] are developed as a reaction to LSTMs to investigate which parts of LSTM were necessary and if memory cells could be made simpler. GRUs are thus simpler than LSTM and need less data to store, as they cannot control how much their memory state is affected by the input. Their memory content is not a special cell state, as it is for LSTM, but the GRU

uses its own output from the previous time step. They have an *update gate* z_t and a *reset gate* r_t. The update gate decides how much the memory is updated. The reset gate determines how much of the memory is forgotten.

The output of the GRU is computed as follows:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tanh(W_h x_t + U_h(r_h \circ h_{t-1}) + b_h)$$

Where $\circ$ is the element-wise Hadamard product. Thus, GRUs use six matrices, three bias vectors and one vector for the cell state, which is two matrices and one vector less than LSTM.

Chung et al. [CGCB14] compared LSTM and GRU, but did not find any significant benefits to use either memory cell over the other. It often depends on the problem at hand.

4. System Overview

Our goal is to generate realistic data-driven gaze animation. Gaze shifts are defined by the motion of the eyes, as well as the central joints, i.e. hips, spine, chest, neck, and head. Therefore, to record human gaze behaviour, we use motion capture to track the joints and eye tracking go gain information about the eyes. With data based on human gaze behaviour, we can train a model to recreate these behaviours as animations. The aim is to generate gaze animations close to human gaze behaviour, that looks more natural than a manually designed procedural animation system. These gaze animations should be generated in real-time.

To recreate the motion present in the data, we use a recurrent neural network generate the animations. Our system predicts the pose of the central joints (hips, spine, neck, head, eyes) in the current frame, based on the location of a gaze target and the character's previous pose. An RNN is trained on motion capture data and video of the face, from which we extract the eye data. Data is recorded for various poses. Separate models are trained for each pose, which can be selected based on the character's current pose at run-time.

Initial data is collected from the existing procedural animation system and later in a motion capture room (see section 5). The setup of the neural network is covered in section 6. Input and output is defined in section 6.1 and the training procedure is described in section 6.2. In section 7 we show how to create a simplified version of the neural network, which can be evaluated faster, for cases where the high quality network would not be necessary.

5. Data Acquisition

In this section, we describe the setups used to record the data to train the neural network. Data was first recorded in Guerrilla Games' proprietary game engine Decima, using Guerrilla's own look-at animation system. This provided us with virtually unlimited training data, that was simpler compared to motion capture data. Scenarios that resulted in useful data for the neural network were later recreated in the motion capture room. Additionally, this enabled us to do some early experimentation with neural network architectures and tuning of hyperparameters. The data from the engine is much cleaner, because the procedural animation system uses

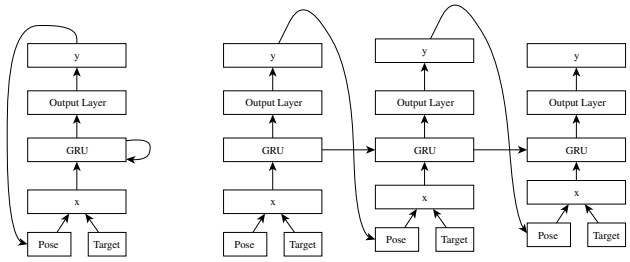

Figure 2: Architecture of the recurrent neural network. On the left the neural network as used during evaluation in the game engine. On the right, the unrolled RNN during training.

simple interpolation functions, which can act immediately. Human gaze behaviour is more complex, due to reaction times and overshoots of the targets. If a model cannot learn the data from the existing animation system, it is not able to learn from the more complex motion capture data. For example, this set a clear minimum number of neurons in the hidden layer, as well as a starting point of values for the learning rate.

The setups that worked in the engine were then recreated in a motion capture room. First, we will describe the incremental process of creating the data recording setups in the engine and why these setups were chosen. This is followed by a detailed description of how the setups were recreated in the motion capture room. Finally, we discuss the extraction process for the eye data.

5.1. Gaze Following

Often, a character in a virtual world needs to follow the avatar of the player with their gaze, when the player passes by. We call this *gaze following* and this applies to targets that slightly move, but do not require large saccades. To recreate this kind of behaviour, we randomly move a target that within a limited space around a character.

In the engine this is accomplished by randomly placing a look-at target in a box of $6m$ wide and deep, and $3m$ high, such that the target can move far away enough from the character. Although a hemisphere around the character would be the ideal shape to sample target positions, we used a box because they are easier to sample. A goal is selected randomly in the box and the target is moved towards the goal over $3.33s$, by linearly interpolating between the original position and the goal. When the target reaches the goal, a new goal is selected and the target is moved to the new goal. While the target is being moved, the character is constantly following the target with its gaze.

We recreate the gaze following setup in motion capture room as follows. A person is recorded in a motion capture room to create gaze following training data. We recorded positions and orientations of all central joints (hips, spine, neck, head), as well as the limbs (shoulders, elbow, wrist, knee, ankle). In addition, we used a head mounted camera to record the face. Since we are working with multi-modal data from two sources (motion capture and video), they have to be synchronised. A clapper with motion track-

Figure 3: Gaze following scene in Decima engine, using the manually designed procedural gaze system. The floating mug is the gaze target.

ers is used to indicate the joint start of the recording. The video is recorded at 60 Hz and is later edited such that the video frame rate is dropped to match the frequency of the motion capture: 29.97 Hz.

To capture gaze following behaviour, we use the end of a pole as the target. The pole is tracked in the motion capture, so the target is a point in the motion capture data. One person will follow the target with their gaze while the pole is moved around them. In order to create a more robust gaze model, the target is moved at various heights and speeds, going behind the person, and overhead. The scenario was repeated for different poses, including standing, sitting, walking and laying down (see Table 1). Many of these poses put additional constraints on the range of movement for gaze, e.g. being unable to rotate from the hips when sitting down.

Action	time (sec)	frames	ratio (%)
Standing	633.33	19000	31.25
Sitting	633.33	19000	31.25
Laying	280	8400	13.82
Walking	480	14400	23.68
Total	2026.67	60800	100

Table 1: Different modes of data used for training.

5.2. Eye Tracking

During the motion capture sessions, the face was recorded with a head mounted camera. OpenFace [BRM16] was used to extract the gaze and blinks from the video footage. Sometimes, OpenFace can lose track of the face, often due to poor illumination of the face, resulting in low confidence classification. When confidence is too low, the data cannot be used for training. A LED light was placed on the head mount to better illuminate the face, resulting in a high confidence rate on the eye tracking for every frame. The extraction process is not precise enough to provide information about the gaze direction of separate eyes. Due to this limitation, data about vergence could not be extracted.

6. RNN for Gaze Animation

The neural network to control gaze animations consists of a recurrent layer, like LSTM or GRU, followed by a feedforward output layer. LSTM and GRU are used as defined above. The output layer has no activation function and is meant to ensure the representation by the recurrent layer is converted into the correct unit vectors. The RNN is presented with the pose of the central joints of the character and a gaze target and it predicts the pose of the character in the next frame. See Figure 2 for an overview of the architecture.

The recurrent layer allows the network to deal with reaction times, overshoots of the target, and large time dependent movements, like turning from the far left to the far right, or vice versa. An RNN is capable of remembering where the motion started and it can remember that the body should rotate along the long path, which can take between 1 and 2 seconds. Although a feedforward network was capable of learning the existing animation system, it was unable to learn properly from the Decima data, because humans' reaction time and overshoots of the target.

6.1. Input and Output

Each frame, the RNN predicts the pose of the character in the current frame, provided by the target and the previous pose of the character. The input vector at frame i is $x_i = t_i, j^r_{i-1}, e^r_{i-1}$, where t_i is the unit vector from the head to the target, relative to the character's root orientation, j^r_{i-1} are the joint rotations of the central joints of the character, i.e. hips, spine, neck, and head, in the current frame. e^r_{i-1} is the forward vector of the eyes from the current frame. The eyes can be represented with only one vector, since they have one less degree of freedom (eyes cannot do a roll rotation). Both forward vector of the eyes and joint orientations are relative to the character's root orientation.

Rotations of joints are represented by six numbers, representing the up and forward vector of each joint. When the up and forward vectors are known, the rotation is completely set. This representation is similar to the work done by Zhang et al. [ZSKS18]. Pavllo et al. [PGA18] use quaternions to represent the rotations, but this requires an additional layer after the output layer to ensure normalisation. We found that quaternions without a normalisation layer during training did not perform well, but using two vectors for rotations without explicit normalisation did give correct results. Thus, to maintain efficiency and simplicity of the model, quaternions were not used. It is possible that quaternions could improve run-time performance by decreasing the length of the input and output vectors, due to their more compact representation.

Only the central joints are used as input. Although the arms and legs could theoretically add more information about the character's pose, and thus the constraints on rotations, arms and legs have been omitted here. In our tests it deteriorated the results, because the RNN could not generalise correctly over the extra joints and found correlations between the arms and legs and the central joints. This led to unpredictable behaviour when running the neural network on a character which had a pose where the arms and legs were in a configuration that is not present in the training data. The output vector is defined as $y_i = j_i^r$, which contains the joint rotations of all the central joints, and is similar to the input vector joint representation.

6.2. Training

The input and output vectors are combined in minibatches of size 64. The RNN is trained on sequences of 120 frames, or four seconds of data. The dataset is mirrored to double the amount of data. Both inputs and outputs are then normalised by subtracting the mean and dividing by the standard deviation.

To predict the next values of rotations of the joints is a regression task. To compute the error between the prediction and the expected value, Mean Squared Error was used:

$$COST(X,Y,\theta) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \theta(X_i))^2$$

where X is the input, Y is the expected output, and θ is a model for predicting Y from X. The ADAM optimiser [KB14] was used to train the model, with an initial learning rate set to 0.0001, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The hidden layer consists of 512 neurons. Training was done in the deep learning framework TensorFlow [ABC*16] and was performed using an Intel Xeon E5-1650 CPU at 3.6 GHz, an Nvidia GTX 1080 GPU, and 64GB of RAM. The model is trained for 50 epochs, taking roughly 12.5 hours to train. The values of the training hyperparameters were selected through a grid search [GBC16] over potential values and selecting the values that resulted in the most natural gaze motions.

The neural network is trained using its own outputs as inputs combined with the target from the captured data, except for the first frame of a training sequence, which is completely extracted from the data. Experiments with teacher forcing [GBC16], where the RNN trains using only the ground truth inputs, shows that it does not benefit the model, thus slowing down training compared to training directly with the RNNs own outputs. Dropout [SHK*14] could not be used in our RNN, since that would disrupt the information flow of the RNN during backpropagation through time. Zaremba et al. [ZSV14] suggests using dropout only on the inputs, but in our case this would still disrupt the information flow, because our RNN uses its own outputs as inputs. Martinez et al. [MBR17] found that an RNN that uses it own output as input results in enough variation in the input to function as regularisation. However, we found that this did not provide enough regularisation to prevent overfitting. When overfitting occurs, the gaze animation appears to shake and jitter. Too much regularisation makes in the model no longer capable of learning all motions and appears stiff and unresponsive.

We experimented with *weight decay*, specifically L^1 and L^2 *regularisation*, also known as *LASSO* and *ridge regression*, respectively [GBC16]. These regularisation techniques put a constraint on the sum of the absolute weight values for L^1 and the sum of squared weights for L^2, defined as

$$L^1(\theta) = \sum_i |w_i|$$
$$L^2(\theta) = \sum_i w_i^2$$

Then, the cost function gets expanded to

$$COST(X,Y,\theta) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \theta(X_i))^2 + \lambda * L^l(\theta), \text{ for } l \in \{1,2\}$$

where λ is a tunable hyperparameter. In our experiments we found that both L^1 and L^2 were able to limit overfitting, eliminating the jitter in the animation. L^1 easily limitted the model too much, causing the network to become unresponsive. L^2 on the other hand was able to remain responsive, although we found that there is a sharp drop off point for responsiveness. We used L^2 regularisation with $\lambda = 8 * 10^{-5}$. Increasing λ to just $1 * 10^{-4}$ already starts to decrease responsiveness.

7. Level of Detail

In many areas of computer graphics, adaptively controlling the level of detail (LOD) is commonplace to keep a steady frame rate. For example, trees or characters far away are rendered with a lower polygon counts and lower resolution textures. As the camera gets farther away from the objects, fewer details are rendered. A similar approach can be taken for animation. Very nuanced motions will not be noticed from afar and can be left out when the camera is far away. For look-at animations, the eyes become too small at a distance for the irises and pupils to be noticed. Controlling the eyes beyond such a distance is unnecessary and can be left out. Some details in the motion of the central joint can also be simplified. The motion capture data shows that there is a specific coupling in the joints between the hips and the chest, and between the neck joints. This makes it possible to remove some details, by only predicting the hips, the chest, and the head, and interpolating the joints in between. This will make the approximation by the neural network less precise, but faster to evaluate.

For the LOD version of the gaze network, the eyes are omitted, as well as seven out of the ten central joints, see Figure 4. Only the hips, chest, and head will be used for the input and output. This means that the input changes from 66 tot 21 and the output from 63 to 18. The number of neurons of the recurrent layer can also be decreased, since less information need to be processed and stored. This decrease in model size decreases training time and evaluation

time. The LOD network is trained separately, with all other hyperparameter the same.

The seven joints that are not predicted by the LOD network are set by interpolating between the hips and chest or chest and head, for the spine and neck, respectively. This is done using a slerp (spherical linear interpolation) operation on the quaternions obtain from the rotation matrices of the predicted joints. These rotation matrices are defined by the predicted forward and up vectors.

The LOD network is trained with 96 hidden units and a learning rate of 0.001. The regularisation coefficient $\lambda = 0.0001$, which is slightly higher than for the complete network, because the number of weights grow quadratically with the number of hidden units. This means that a larger model requires a lower regularisation coefficient. Similar to the complete model, the hyperparameters were selected through a grid search of potential hyperparameters values. Other than that, all other hyperparameters are unchanged for the LOD network compared to the complete network.

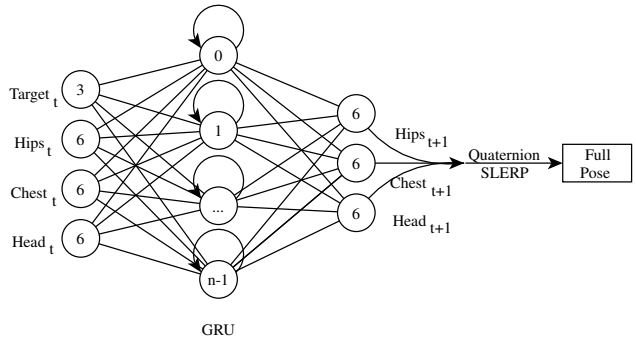

Figure 4: Architecture of the LOD network.

8. Results

After training, the neural network is exported from TensorFlow as a text file, then parsed and loaded into the game engine and evaluated using our own implementation of forward propagation. The recurrent layer stores its memory, along with the weight matrices and bias vectors, and can update it each frame. The gaze animation is applied after all other animations and it keeps track of its previous output, which is to be used as input in the next frame. A gaze target is provided from game code and is converted to a unit vector, that starts at the character's head, and is rotated relative to the character's root orientation.

8.1. Evaluation

The neural network is capable of more realistic movement than the manually designed animation system for gaze, especially in more extreme gaze directions, like looking directly behind or above the character (see Figure 5 and 1). The RNN can also adopt more constrained poses, like sitting or laying down (see Figure 6 and 7), which can be difficult to create a robust procedural system for. If the model is not complex enough, it will start to underfit. This results in the head rotating beyond its limits and shaking movements, because the approximation of the motion is too coarse to move smoothly. Underfitting also happens when the training sequences were not long enough. Four seconds seem to be the minimum duration, such that the largest motion can be correctly learned. When trained on sequences of 1 or 2 seconds the head would rotate backwards in some situations, because the RNN had never seen the complete motion of reaching the rotation constraint on the far left and then moving all the way to the utmost constraint on the right. With sufficiently long training sequences, these issues do not occur.

Models can be trained to learn various poses and if a new pose needs to be added, that situation can be recreated in a motion capture room for additional data, and a new model can be trained for that new situation. This is easier than extending an existing procedural system, where internal dependencies can make the system increasingly complex to maintain and extend. Compared to the existing procedural animation system, the neural network is capable of more variety in its motions. This is seen in slight delays, due to reaction time, overshoots of the target if the target suddenly changes direction, because the network has also learned to make predictions about the target.

Between LSTM and GRU we could not find one that performed notably better visually. They were both able to approximate the data, but the GRU gets the preference here, because it uses two fewer matrices. Thus, GRU has fewer variables to train, resulting in lower training times, as well as faster evaluation of the model.

Compared to the motion capture, there are still some differences. This is largely due to the fact that the neural network can only control the central joints of the skeleton. When people stand in a relaxed pose and follow the target with their gaze, they will displace their weight depending on their gaze direction: if one is looking to the left, they tend to lean to the left, unless the target is close by, in which case people try to keep their distance by leaning in the opposite direction. Because the neural network can only control the central joint and not the legs, this behaviour cannot be mimicked by the system. It is possible that the complete skeleton can be successfully controlled if enough data is present, but we did not have enough data to test this hypothesis (see section 6.1).

The RNN can be evaluated in $0.585ms$ and requires $3.52MB$ of memory, making it suitable for real-time animation. The LOD network runs in $60\mu s$ and requires $0.20MB$, which would enable this technique to be used for a large number of characters in a scene.

8.2. Comparison

Selecting a neural network architecture for animation is a non-trivial task. For locomotion there have been several papers, each using a different architecture [FLFM15, **?**, HKS17, ZSKS18]. We compare our architecture with two of these approaches that also use RNNs and show why our method is preferred to the neural architectures used for locomotion. We also compare our method to a feedforward network and a stacked RNN, to show that the recurrent layer is necessary and that adding more recurrent layers does not improve performance over a single recurrent layer.

In total, we compare five different neural architectures: a feedforward network, our RNN with one recurrent layer, a stacked RNN with two layers, a RNN with residual connections proposed

Figure 5: Constraints are learned. Here, the target moves behind the back of the character, which requires a rotation from the far right to the far left.

Figure 6: The RNN can also perform gaze while sitting, adding constraints to the hips, limiting the reach of the gaze animation.

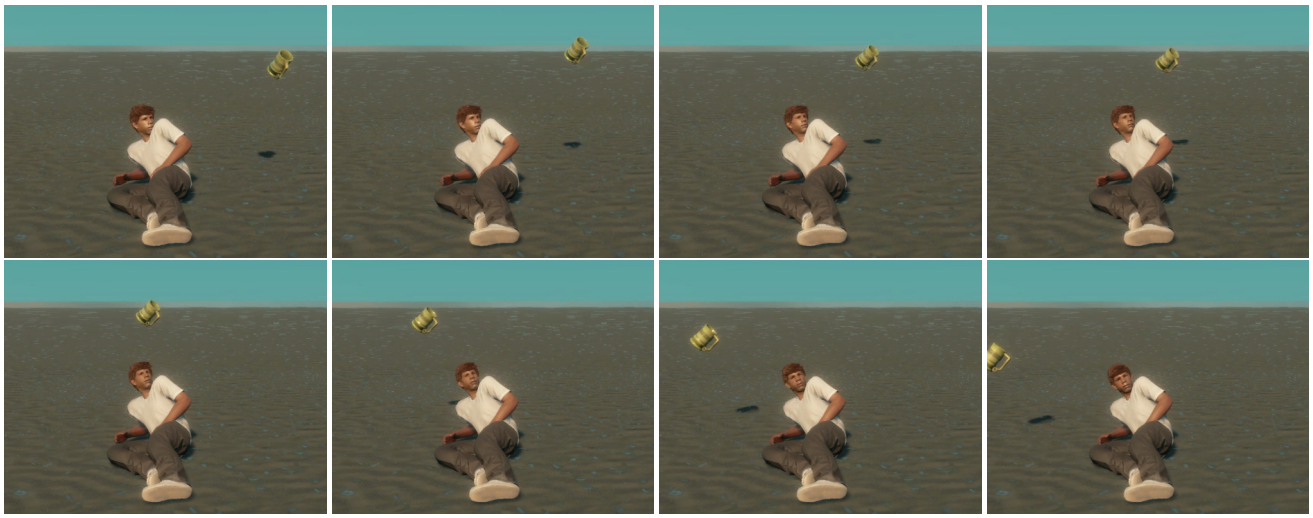

Figure 7: When laying down, most joints are constrained except for the head. The target cannot be completely followed anymore.

by Martinez et al. [MBR17], and an Encoder-Recurrent-Decoder (ERD) proposed by Fragkiadaki et al. [FLFM15]. Their architectural designs can be seen in Figure 8. The ERD uses a feedforward encoding layer before and a feedforward decoding layer after the recurrent layer. These encoding layers enables the ERD to map the input into an encoded state that is better suited for the recurrent layer and the decoder can map it out of the encoded state. Both layers have half the number of neurons compared to the recurrent layer and both use ReLU activations, similar to the setup used by Fragkiadaki et al. for pose prediction. Residual connections, sometimes also called skip connections, link the input directly to the output. In our case, they add the input pose to the predicted pose, which changes the learning problem from learning complete poses to learning the differences in poses. The feedforward network has two feedforward layers with ReLU activation, followed by a linear output layer. All architectures are created to have the same memory footprint as our proposed method. Each network is trained for 50 epochs, with a batch size of 32, and a sequence length of 120 frames, except for the feedforward network, which can only train on single frames. The results of the comparison can be found in Table 2.

The feedforward network is not able to learn properly from the motion capture data, which can be seen from its error rate. Due to its lack of temporal information, the feedforward network generates rather static animations that looks robotic. Experiments by adding the angular velocities of each joint did not show any significant improvements. The model is capable of gaze following for some time, but cannot correct its own mistakes like the RNNs can. Eventually, it generates too extreme values, exploding the joint rotations until numerical errors puts it in a unresolvable state (see appendix).

The RNN with residual connections [MBR17] was not able to get the same level of accuracy as the other RNN models. Learning the difference between poses does not improve learning results in gaze animation. The residual network quickly looses track of the adjustments it has to make to the pose. This in turn places the character in poses the network has never seen before, which it has never learned to recover from. Soon after initialisation, the character is twisted in impossible poses, making this approach too unstable.

The ERD [FLFM15] give similar results to our method, but does not improve on them. However, the added encoder and decoder layers do increase training time, evaluation time, and the number of hyperparameters. Since there is no improvement in animation quality, there is no benefit in adding the encoder and decoder layers. Because our goal is to create a model in real-time virtual environments, the run-time performance of our method is preferred. The same arguments used for the ERD are applicable to the stacked RNN. Adding a recurrent layer to the network does not improve animation quality, but does increase training time and evaluation time.

Model	**Layer size**	Error score	Training time (50 epochs)	Run time (us)
Feedforward	896, 896	5.4499	10 min	500
RNN	512	0.1264	1:10 h	585
Stacked	312, 312	0.1373	1:30 h	658
Residual	512	0.1824	1:15 h	580
ERD	212, 424, 212	0.1162	2:15 h	700

Table 2: Training results of the comparison.

8.3. User Study

To evaluate the perceived naturalness [VWVBE*10] of the gaze animation produced by the neural network, we conducted a user study. We compared our proposed model and its LOD version against the procedural approach in Decima and the ground-truth motion capture data. The motion capture data was recorded specifically for the user study and was not used to train the model.

8.3.1. Survey Setup

Our hypothesis was that (1) the LOD network would be preferred to the procedural system, (2) the complete neural network would be preferred to the LOD network, (3) the ground truth motion capture would preferred to the neural network. More formally,

$$\text{procedural} < \text{LOD RNN} < \text{complete RNN} < \text{motion capture}$$

The users were shown 18 videos in total. Per video, two out of the four techniques were shown side-by-side, resulting in 6 videos per pose. The user is asked to select the video that they perceive as most natural. Each video is roughly 10s long. The comparison videos were grouped by pose (so the participants could easily determine how far they were) and were presented in a random order per pose.

The survey was send out to game industry professionals within Guerrilla Games. First, the participants were asked which department they belonged to, e.g. art, animation, programming, etc. We were particularly interested in differences in answers between animators and non-animators, since we expected animators to have a more critical eye on this topic. A total of 73 people participated. 16 of them were animators.

8.3.2. Results

The results of the user study can be found in Figure 9. For the values of the statistical analysis, see the appendix. In all cases, there is a significant preference over all other techniques compared to the old procedural system, which is in line with our hypothesis. The original motion capture (mocap in Figure 9) was not significantly preferred to the complete RNN, when sitting. For the sitting pose, the motion capture is often perceived as less natural than the complete RNN and LOD network, which can be attributed to a difference in poses between the motion capture and the other techniques. When laying down, the motion capture is significantly preferred over all other techniques. This can be contributed to the motion of the arm the character is resting on. This arm is still on the ground in the motion capture, while it can still move a little bit in the other techniques. The arm movement makes the other techniques seem less natural.

Surprisingly, the LOD network was preferred to the complete RNN, in all poses except the sitting. When sitting, the results were split down the middle. The motions generated by the LOD network

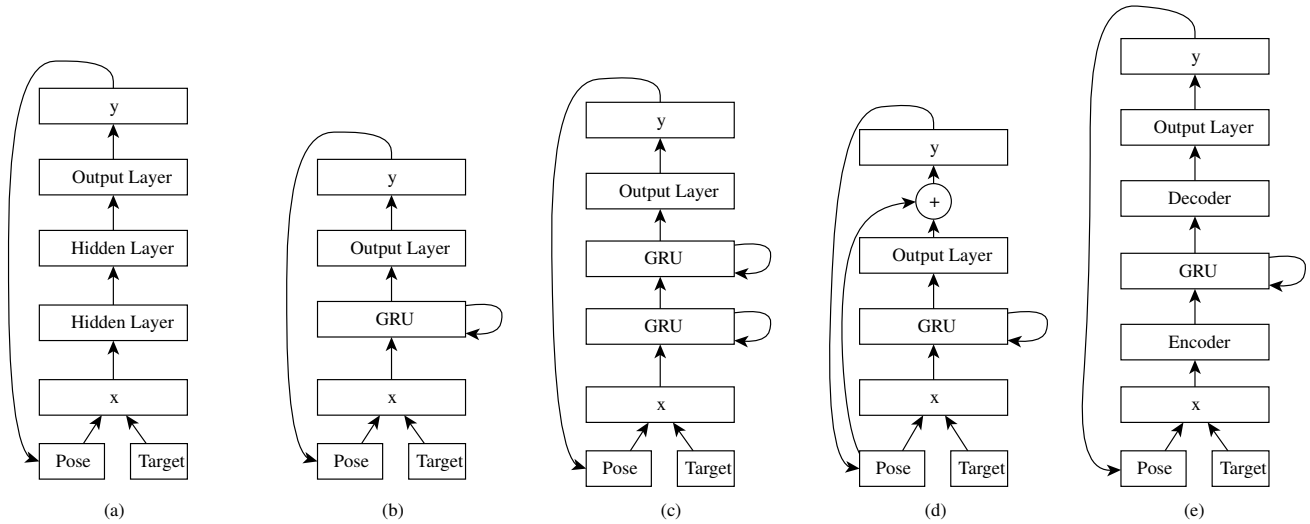

Figure 8: The various architectures used for our comparison: (a) feedforward network, (b) GRU with output layer, (c) stacked GRU with output layer, (d) GRU with residual connections, (e) ERD network.

could appear smoother, because it is trained on a simpler task, i.e. it can predict more precisely. Together with the quaternion interpolation, jitter can be smoothed away, which might still occur on the complete RNN, which predicts seven more joints. Not only is the LOD network simpler and faster, it also is perceived as more natural. Animators' responses were in all cases in line with the results from all participants.

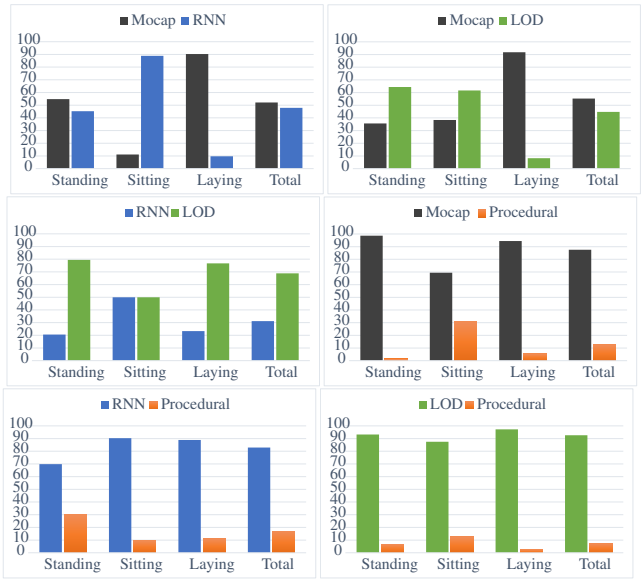

Figure 9: Results of the user study. Participant were asked for each pair of techniques which one they preferred.

9. Limitations and Future Work

It is possible to find undesired motions when the neural network is confronted with targets that are not present in the training data. This occurs for example when the target moves past the head very closely, which can result in a high angular velocity on the target vector. Although this could be improved by more training data which capture these scenarios, the issue can also be alleviated by preventing the gaze target to move that close to the character.

This research has focussed on gaze following behaviour, but the model struggles with large saccades, since that was not present in the training data. It would not be difficult to train the model with such data, but we did have difficulty recording such behaviour reliably. We have tried one setup for large saccades, where gaze targets were placed in a dome around the gazing person. Each target was a LED light and a buzzer, which could be controlled through an Arduino, as well as a motion capture tracker, so each target was also present in the motion capture data. A script in the Arduino would turn on the targets one by one in a randomised order. Unfortunately, people could not effectively determine where the sound was coming from, which turned the gaze behaviour into search behaviour. This made for very noisy data, that was not useful for training. If large saccadic data could be obtained, we do expect that this could be learned, as this was the case for the data from the procedural animation system.

The neural network can control the eyes, although results on the eyes were inadequate. This is mostly due to the quality of the eye data in the training data. The consistency of the gaze direction extraction (and blink detection) of OpenFace is not good enough for the purpose of this research. However, as the technique to extract these features from video becomes better, the quality for prediction by a neural network also improves.

Sometimes the naturalness of the gaze animation is decreased by

movement in joints not directly controlled by the neural network. For example, the shoulders move along with the gaze animations when the character is laying down, which also moves the supporting arm. Similar artifacts in the kinematic linking can be found in the feet when the hips makes a roll rotation. This could be solved in other part of the animation system, e.g. extended foot locking, but this is beyond the scope of this thesis.

The proposed gaze network is trained to provide gaze behaviour in situations where characters are not particularly interacting with other characters. Gaze behaviour changes when people are in a conversation with each other. We do consider this another problem and beyond the scope of this research, but it would be interesting to see the performance of the proposed method on conversational gaze scenarios.

10. Conclusion

We propose a data-driven method for creating gaze animations by using a recurrent neural network (RNN). Our method is capable of learning gaze animations in various poses from multi-modal data from motion capture and a head-mounted camera. We describe how the training data was recorded in the motion capture room. The RNN is fast enough for real-time virtual environments. To make our method applicable to even large groups of virtual characters, we also designed an RNN on a lower level of detail (LOD), for characters that are further away from the view point. We have performed a user study among game industry professionals, which shows that both the complete RNN and LOD network improve naturalness of the gaze animations, compare to the procedural gaze animation. The LOD network is perceived as more natural than the complete model, making the LOD network a great approach for real-time gaming, even in scenarios with many characters.

11. Acknowledgments

We would like to thank Guerrilla Games and Sony Interactive Entertainment for making this research possible. A special thanks goes to the AI team for their help and support, Bart Wijsman for his help with the motion capture data and animations, Kevin Quaid for his help with the head-mounted camera, Thijs Kruithof for integrating the neural networks into Decima, and Ana Bartuba for formatting and distributing the user study.

References

[ABC*16] ABADI M., BARHAM P., CHEN J., CHEN Z., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., IRVING G., ISARD M., ET AL.: Tensorflow: A system for large-scale machine learning. In *OSDI* (2016), vol. 16, pp. 265–283. 8

[BRM16] BALTRUŠAITIS T., ROBINSON P., MORENCY L.-P.: Openface: an open source facial behavior analysis toolkit. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on* (2016), IEEE, pp. 1–10. 7

[BSPP08] BRKIC M., SMID K., PEJSA T., PANDZIC I. S.: Towards natural head movement of autonomous speaker agent. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems* (2008), Springer, pp. 73–80. 2

[CGCB14] CHUNG J., GULCEHRE C., CHO K., BENGIO Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014). 5, 6

[CGV09] CAFARO A., GAITO R., VILHJÁLMSSON H. H.: Animating idle gaze in public places. In *IVA* (2009), Springer, pp. 250–256. 3

[FLFM15] FRAGKIADAKI K., LEVINE S., FELSEN P., MALIK J.: Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 4346–4354. 1, 4, 9, 11

[GBC16] GOODFELLOW I., BENGIO Y., COURVILLE A.: *Deep Learning*. MIT Press, 2016. "http://www.deeplearningbook.org". 4, 5, 8

[HKS17] HOLDEN D., KOMURA T., SAITO J.: Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 42. 1, 4, 9

[HS97] HOCHREITER S., SCHMIDHUBER J.: Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780. 5

[HSK16] HOLDEN D., SAITO J., KOMURA T.: A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 138. 4

[KB01] KHULLAR S. C., BADLER N. I.: Where to look? automating attending behaviors of virtual human characters. *Autonomous Agents and Multi-Agent Systems 4*, 1-2 (2001), 9–23. 3

[KB14] KINGMA D., BA J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 5, 8

[LBB02] LEE S. P., BADLER J. B., BADLER N. I.: Eyes alive. In *ACM Transactions on Graphics (TOG)* (2002), vol. 21, ACM, pp. 637–644. 3

[LBH15] LECUN Y., BENGIO Y., HINTON G.: Deep learning. *Nature 521*, 7553 (2015), 436–444. 5

[LKA*17] LAINE S., KARRAS T., AILA T., HERVA A., SAITO S., YU R., LI H., LEHTINEN J.: Production-level facial performance capture using deep convolutional neural networks. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2017), ACM, p. 10. 3

[LM10] LANCE B., MARSELLA S.: The expressive gaze model: Using gaze to express emotion. *IEEE computer graphics and applications 30*, 4 (2010), 62–73. 2

[LMD12] LE B. H., MA X., DENG Z.: Live speech driven head-and-eye motion generators. *IEEE transactions on visualization and computer graphics 18*, 11 (2012), 1902–1914. 3

[LMK04] LANCE B., MARSELLA S., KOIZUMI D.: Towards expressive gaze manner in embodied virtual agents. In *AAMAS workshop on empathic agents* (2004), pp. 194–201. 2

[LZ15] LEIGH R. J., ZEE D. S.: *The neurology of eye movements*, vol. 90. Oxford University Press, USA, 2015. 2

[MBR17] MARTINEZ J., BLACK M. J., ROMERO J.: On human motion prediction using recurrent neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), IEEE, pp. 4674–4683. 1, 4, 8, 11

[PALvdP18] PENG X. B., ABBEEL P., LEVINE S., VAN DE PANNE M.: Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (Proc. SIGGRAPH 2018 - to appear) 37*, 4 (2018). 1, 3

[PBYVDP17] PENG X. B., BERSETH G., YIN K., VAN DE PANNE M.: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 41. 1, 3

[PGA18] PAVLLO D., GRANGIER D., AULI M.: Quaternet: A quaternion-based recurrent model for human motion. *arXiv preprint arXiv:1805.06485* (2018). 1, 8

[PGCB13] PASCANU R., GULCEHRE C., CHO K., BENGIO Y.: How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026* (2013). 5

[PMG13] PEJSA T., MUTLU B., GLEICHER M.: Stylized and performative gaze for character animation. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 143–152. 3

[PRMG16] PEJSA T., RAKITA D., MUTLU B., GLEICHER M.: Authoring directed gaze for full-body motion capture. *ACM Transactions on Graphics (TOG) 35*, 6 (2016), 161. 3

[PVZ08] PROCTOR R. W., VAN ZANDT T.: Human factors in simple and complex systems. 2

[RAB*14] RUHLAND K., ANDRIST S., BADLER J., PETERS C., BADLER N., GLEICHER M., MUTLU B., MCDONNELL R.: Look me in the eyes: A survey of eye and gaze animation for virtual agents and artificial systems. In *Eurographics State-of-the-Art Report* (2014), pp. 69–91. 2

[SHK*14] SRIVASTAVA N., HINTON G. E., KRIZHEVSKY A., SUTSKEVER I., SALAKHUTDINOV R.: Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research 15*, 1 (2014), 1929–1958. 5, 8

[TKY*17] TAYLOR S., KIM T., YUE Y., MAHLER M., KRAHE J., RODRIGUEZ A. G., HODGINS J., MATTHEWS I.: A deep learning approach for generalized speech animation. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 93. 3

[VWVBE*10] VAN WELBERGEN H., VAN BASTEN B. J., EGGES A., RUTTKAY Z. M., OVERMARS M. H.: Real time animation of virtual humans: A trade-off between naturalness and control. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 2530–2554. 11

[ZSKS18] ZHANG H., STARKE W., KOMURA T., SAITO J.: Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics 37*, 4 (3 2018). 1, 4, 8, 9

[ZSV14] ZAREMBA W., SUTSKEVER I., VINYALS O.: Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* (2014). 8

Appendix A: Motion Capture

Figure 10: Images of the motion capture recording in a standing pose, with the head-mounted camera footage in the top-right corner.

Figure 11: Images of the motion capture recording in a sitting pose, with the head-mounted camera footage in the top-right corner.

Appendix B: User Study Survey

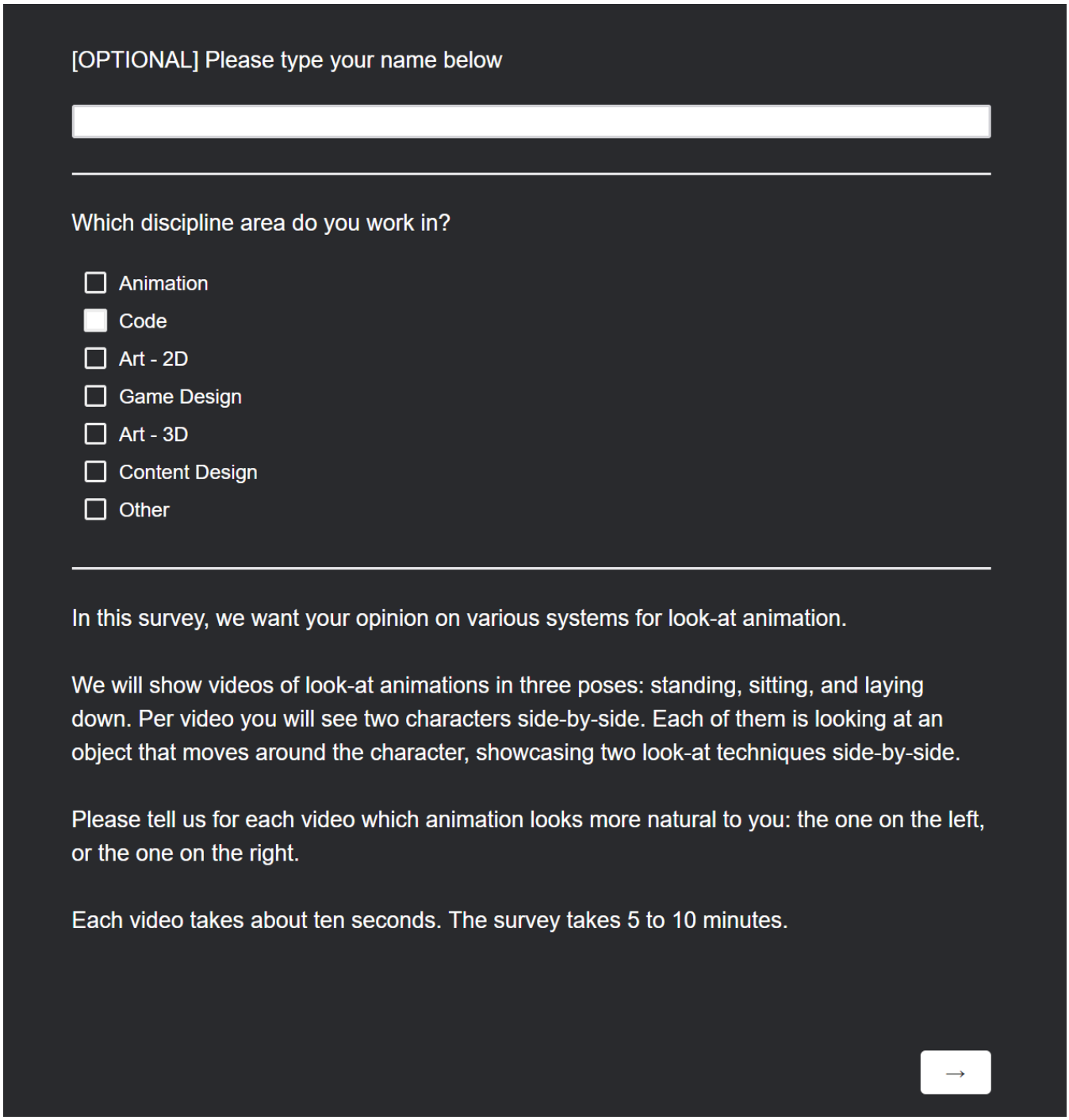

Figure 12: Introduction page to the survey.

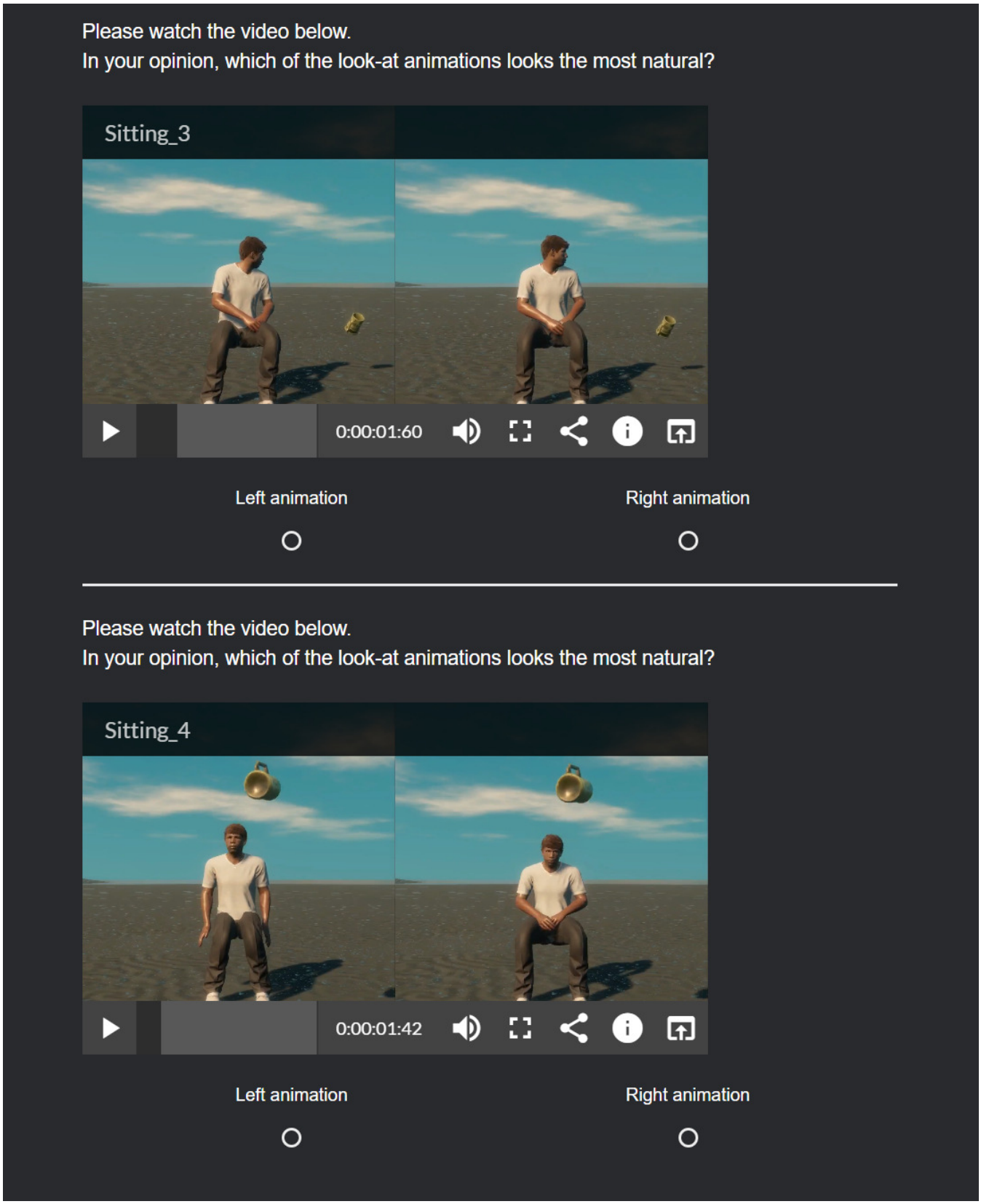

Figure 13: Part of the questions on the sitting pose.

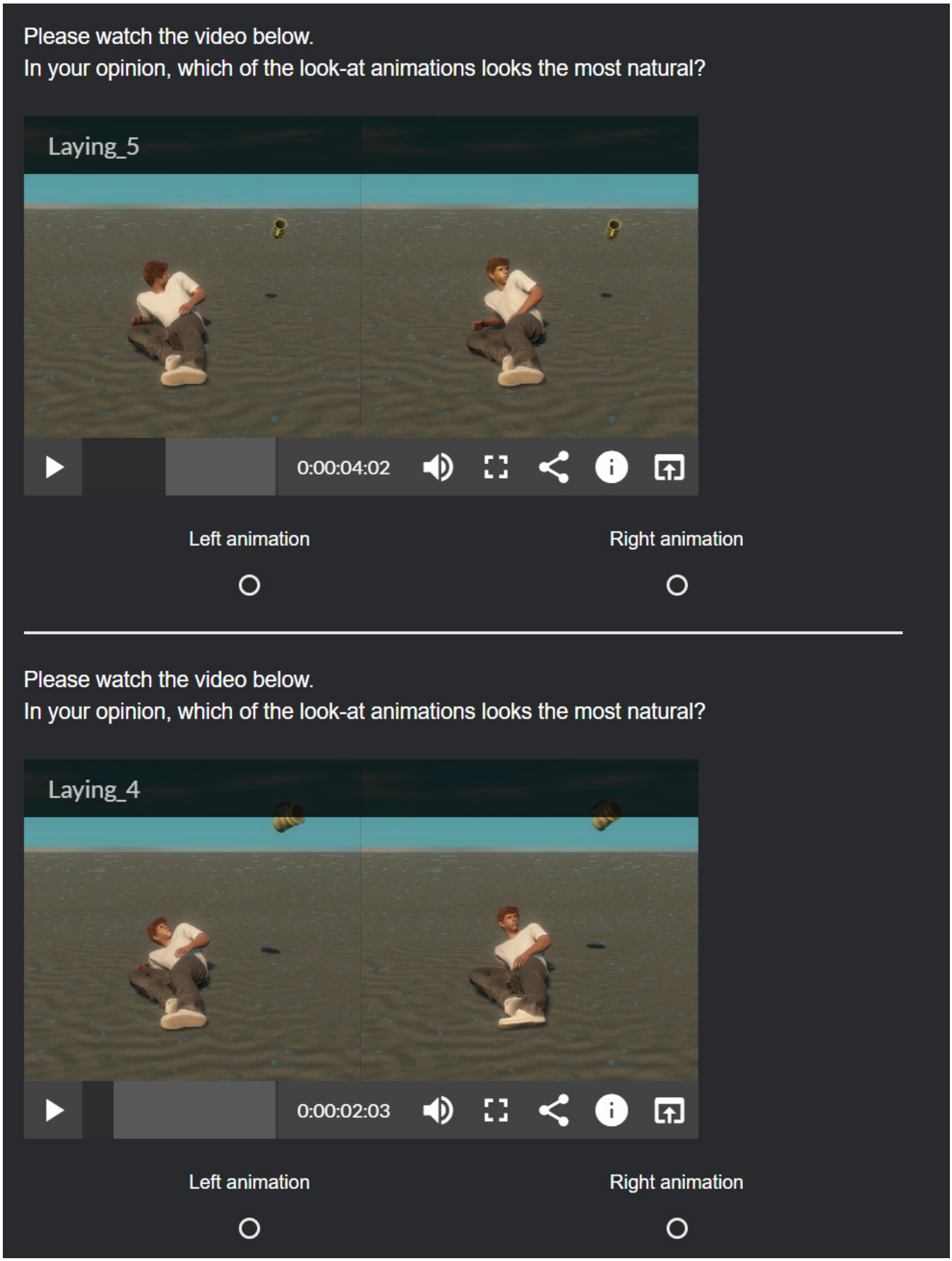

Figure 14: Part of the questions on the laying pose.

Results User Study - All

* marks significant results.

Standing	Left	Right	χ^2	p-value
Mocap, RNN	40	33	0.67	0.4795
Mocap, LOD	26	47	6.04	0.0192*
Mocap, Procedural	72	1	69.05	<0.0001*
RNN, LOD	15	58	25.33	<0.0001*
RNN, Procedural	51	22	11.52	0.001*
LOD, Procedural	68	5	54.37	<0.0001*

Table 3: All preference values for the standing pose, with χ^2 and p-values.

Sitting	Left	Right	χ^2	p-value
Mocap, RNN	8	64	43.56	<0.0001*
Mocap, LOD	28	45	3.96	0.0614
Mocap, Procedural	50	22	10.12	0.0015*
RNN, LOD	36	36	0	1
RNN, Procedural	65	7	46.72	<0.0001*
LOD, Procedural	63	9	40.5	<0.0001*

Table 4: All preference values for the sitting pose, with χ^2 and p-values.

Laying	Left	Right	χ^2	p-value
Mocap, RNN	65	7	46.72	<0.0001*
Mocap, LOD	67	6	50.97	<0.0001*
Mocap, Procedural	68	4	56.89	<0.0001*
RNN, LOD	17	56	20.84	<0.0001*
RNN, Procedural	64	8	43.56	<0.0001*
LOD, Procedural	70	2	62.34	<0.0001*

Table 5: All preference values for the laying pose, with χ^2 and p-values.

All Poses	Left	Right	χ^2	p-value
Mocap, RNN	113	104	0.3	0.5839
Mocap, LOD	121	98	2.42	0.1362
Mocap, Procedural	190	27	122.44	<0.0001*
RNN, LOD	68	150	30.84	<0.0001*
RNN, Procedural	180	37	94.24	<0.0001*
LOD, Procedural	201	16	156.02	<0.0001*

Table 6: All preference values for all poses combined, with χ^2 and p-values.

Results User Study - Animators

* marks significant results.

Standing	Left	Right	χ^2	p-value
Mocap, RNN	11	5	2.25	0.2117
Mocap, LOD	7	9	0.25	0.8065
Mocap, Procedural	0	16	16	0.0002*
RNN, LOD	7	9	0.25	0.8065
RNN, Procedural	11	5	2.25	0.2117
LOD, Procedural	16	0	16	0.0002*

Table 7: Animators' preference values for the standing pose, with χ^2 and p-values.

Sitting	Left	Right	χ^2	p-value
Mocap, RNN	1	15	12.25	0.0012*
Mocap, LOD	4	12	4	0.0802
Mocap, Procedural	4	12	4	0.0802
RNN, LOD	8	8	0	1
RNN, Procedural	16	0	16	0.0002*
LOD, Procedural	15	1	12.25	0.0012

Table 8: Animators' preference values for the sitting pose, with χ^2 and p-values.

Laying	Left	Right	χ^2	p-value
Mocap, RNN	15	1	12.25	0.0012*
Mocap, LOD	14	2	9	0.006
Mocap, Procedural	15	1	12.25	0.0012*
RNN, LOD	3	13	6.25	0.0245*
RNN, Procedural	14	2	9	0.006
LOD, Procedural	15	1	12.25	0.0012*

Table 9: Animators' preference values for the laying pose, with χ^2 and p-values.

All Poses	Left	Right	χ^2	p-value
Mocap, RNN	27	21	0.75	0.4708
Mocap, LOD	25	23	0.08	0.8875
Mocap, Procedural	19	29	2.08	0.1949
RNN, LOD	18	30	3	0.1124
RNN, Procedural	41	7	24.08	<0.0001*
LOD, Procedural	46	2	40.33	<0.0001*

Table 10: Animators' preference values for all poses, with χ^2 and p-values.

Appendix C: Comparison

Figure 15: The feedforward network's generated motion is stale and eventually breaks down.

Figure 16: The stacked RNN works as smooth as our method, but is more expensive to train and evaluate.

Figure 17: The residual network has stability issues.

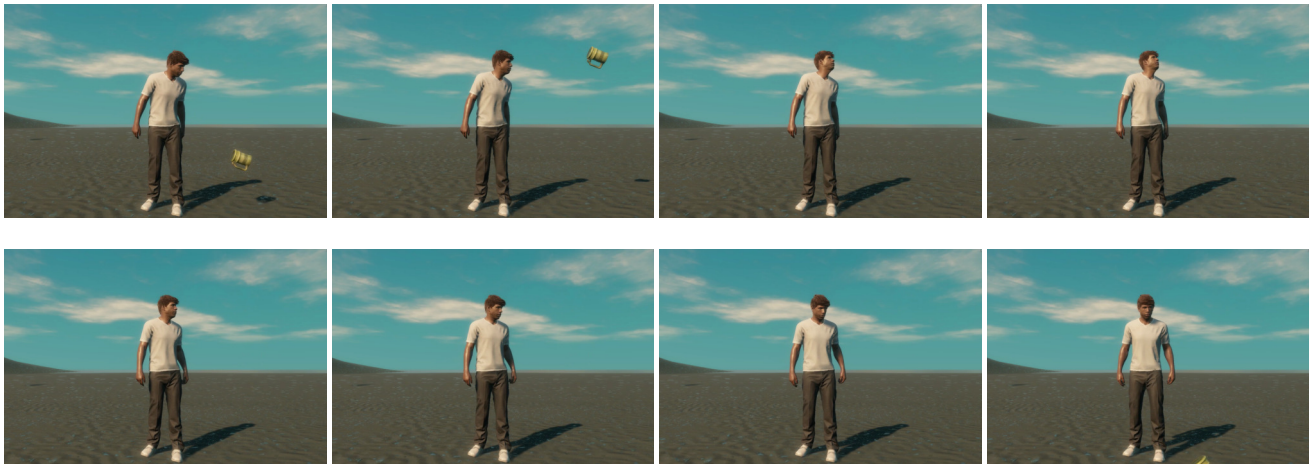

Figure 18: The ERD works as smooth as our method, but is more expensive to train and evaluate.

Appendix D: Complete Network Examples

Figure 19: Occasionally, the complete RNN overgeneralises and tries to rotate the head backwards. Other poses do not suffer from this artifact, because the head cannot rotate to 180 degrees left and right in other poses. Since the joint rotations are predicted in character space, there is no difference between a 180 degree left rotation and a 180 degree right rotation.

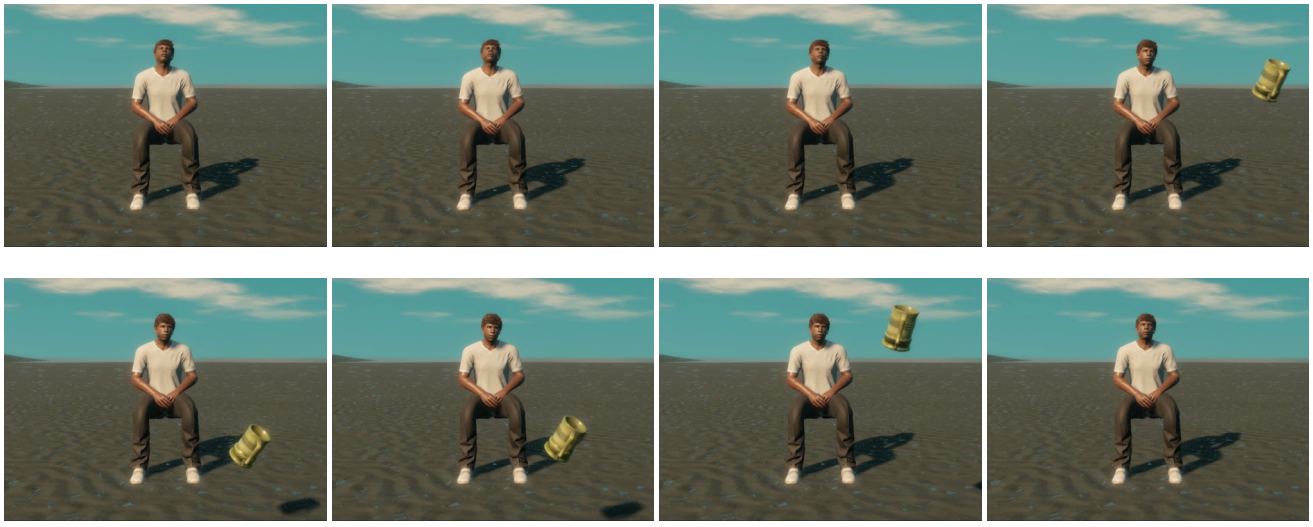

Figure 20

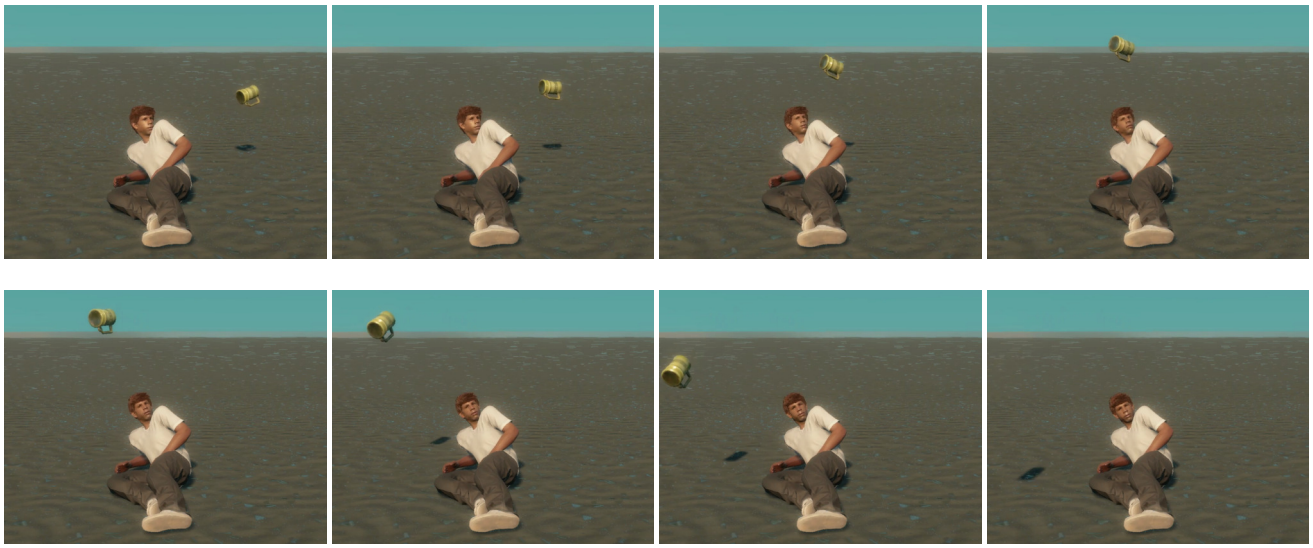

Figure 21: The supporting arm still slightly moves, due to the kinematic linking. This lowers the perceived naturalness of the animation, compared to the motion capture.

Appendix E: LOD Network Examples

Figure 22: The LOD network is trained on a simpler task than the complete RNN. Only the hips, chest, and head are predicted, while the spine and neck are interpolated. The LOD network has learned that it cannot turn the backwards and needs to turn forwards to follow a target moving behind the character.

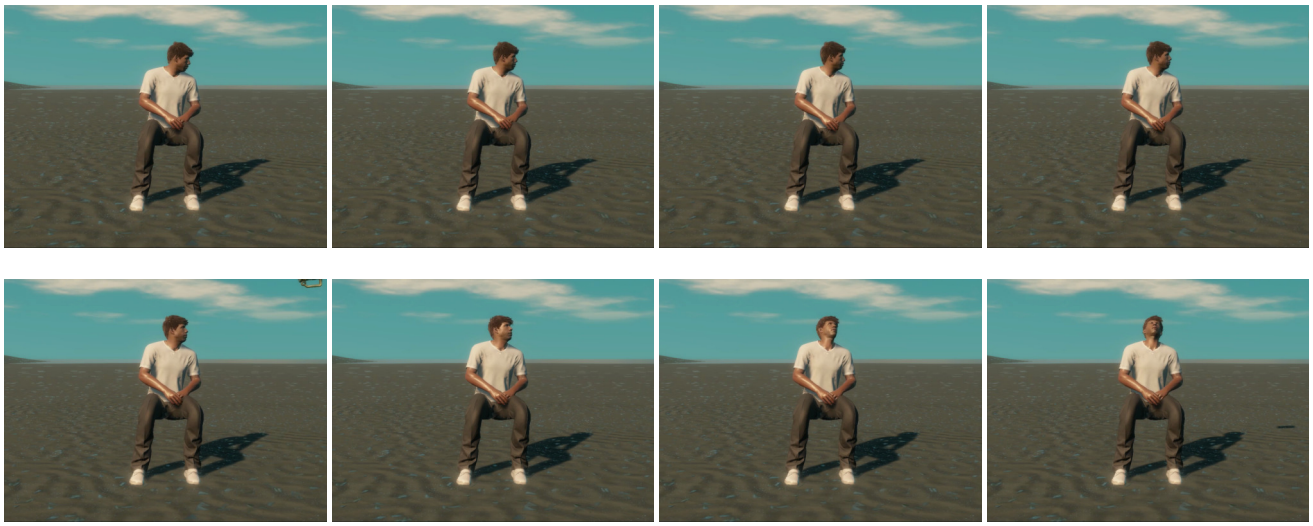

Figure 23: When sitting, the hips can no longer rotate. The LOD network has learned this constraint from the data.

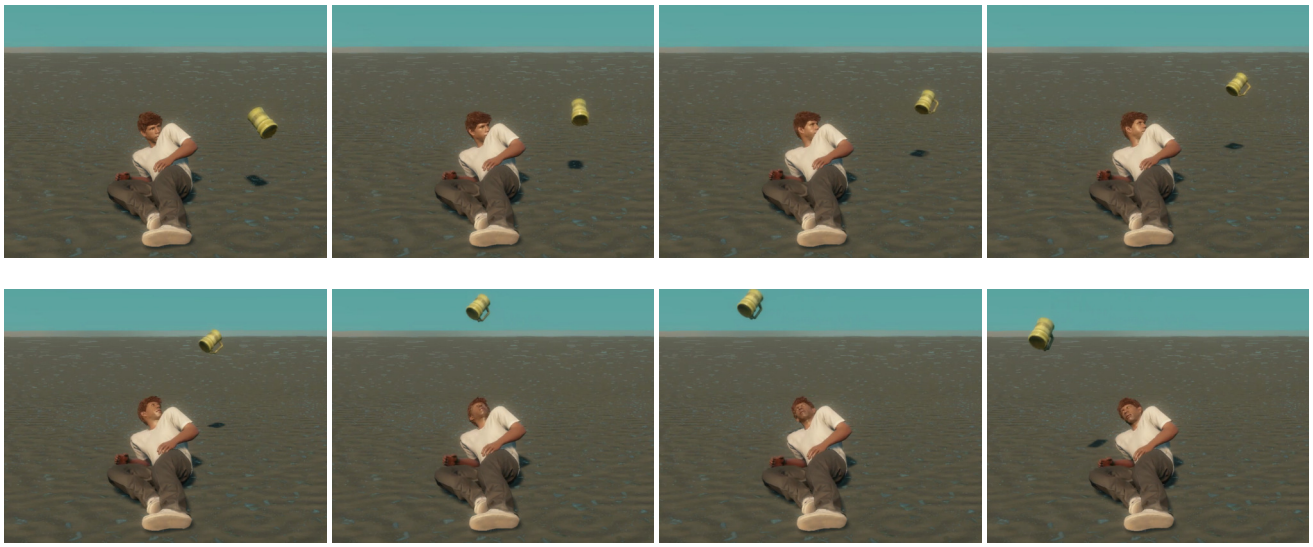

Figure 24: The LOD network mostly turns from the head when laying down, as hips to chest are mostly constraint.