Abstract

Temporal IK: Data-Driven Pose Estimation for Virtual Reality

by

James Lin

Master of Science in Electrical Engineering and Computer Science in

University of California, Berkeley

Professor James O'Brien, Chair

High-quality human avatars are an important part of compelling virtual reality (VR) experiences. Animating an avatar to match the movement of its user, however, is a fundamentally difficult task, as most VR systems only track the user's head and hands, leaving the rest of the body undetermined. In this report, we introduce Temporal IK, a data-driven approach to predicting full-body poses from standard VR headset and controller inputs. We describe a recurrent neural network that, when given a sequence of positions and rotations from VR tracked objects, predicts the corresponding full-body poses in a manner that exploits the temporal consistency of human motion. To train and evaluate this model, we recorded several hours of motion capture data of subjects using VR. The model is integrated into an end-to-end solution within Unity, a popular game engine, for ease of use. Our model is found to do well in generating natural looking motion in the upper body.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to first thank Prof. James O'Brien, whom as my advisor gave me huge amounts of support and advice over the course of this project. Next, I would like to thank Kasra Ferdowsi, my partner on this project, for his work in developing the BVH Visualizer and his assistance with the motion capture recording sessions. I would also like to thank Dr. Allen Yang, both for his feedback on this paper and for allowing me to use his motion capture studio. Without these people, this project would not have happened.

I also thank Paxtan Laker, Sophia Batchelor, and Austin Davis for lending me their Oculus accounts during the motion capture sessions so I could record data on different applications, as well as David McPherson and Justin Yim for their help in getting me set up to use the motion capture equipment. I thank Aurick Zhou, for entertaining my many questions about neural networks and Tensorflow. Credit is also due to the Virtual Reality @ Berkeley club as a whole, for inspiring me to work on this project in the first place.

Lastly, I thank my family, for their continuous support towards my higher education and far more.

Chapter 1

Introduction

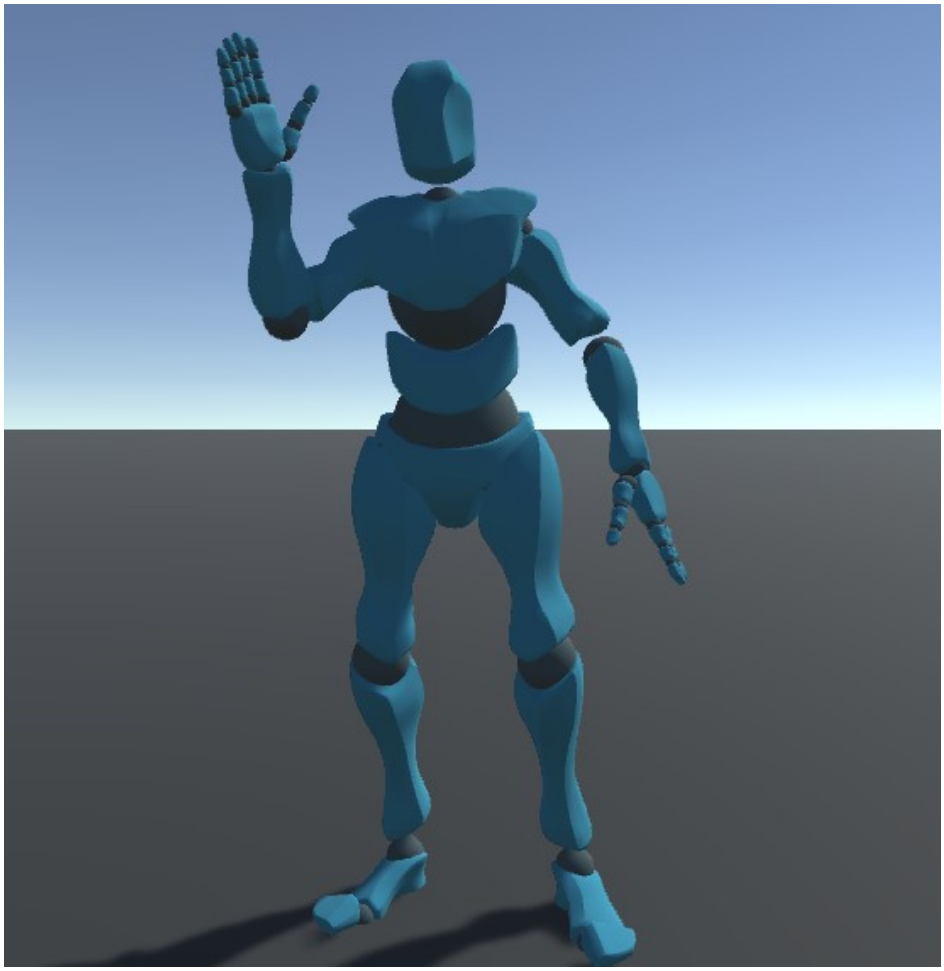

Figure 1.1: Visualization of a predicted pose from Temporal IK.

1.1 Motivations

Nearly all virtual reality (VR) applications are designed to provide an immersive experience for the user - ideally to engage them to the degree where they forget that they've put on a headset. Because of this goal, VR experiences, even more so than traditional mediums, benefit from having high levels of fidelity and consistency within the environment. One component of every VR experience is human avatar representation; that is, the appearance of humans within a virtual environment. Users that can accept their avatar as a proper representation of themselves are more engaged with the overall experience. A well-implemented avatar system can strengthen social interactions between players or the user's sense of self within a virtual environment [25]. Because of these reasons, major players in the VR field such as Oculus and Magic Leap have prioritized research in this area [10] [28]. In this report, we address one of the critical problems in creating avatars for virtual reality applications: estimating human poses from VR inputs.

With most VR devices, the user's head and hands are tracked using a head-mounted display (HMD) and controllers, each with 6 degrees of freedom (DOF). The task of pose estimation in this context is to use this information to predict the position and rotation of all segments in a kinematic skeleton representing the user's body.

There are a number of factors that make it difficult for VR avatar systems to solve this problem. For example, the pose estimation algorithm must be fast enough to run at the high framerates that VR demands. The resulting animation must also accurately imitate natural movement. Animations that are significantly off risk falling into the "uncanny valley", a spectrum of pseudo-realism that evokes fear or disgust from people. Finally, and most crucially, the task of predicting a full-body pose from the inputs of a standard VR system is fundamentally under-determined. Many joints of the body, including the elbows, hips, and lower body, are not constrained by knowing where the head and hands are, which means multiple valid poses can map to the same input values.

Most VR applications currently do not use full-body avatars because of the problems stated above. Instead, many provide an abstract representation of the body with disconnected, floating head and hands. This representation makes pose estimation easy, but does little to improve the quality of the experience. Some applications, such as Ready at Dawn's game Echo Arena, use inverse kinematics (IK) to control the body [7]; however this approach leads to unnatural movements and poses as shown in Figure 1.2. A few applications support additional hardware. For example, the social platform VRChat allows users to wear tracked controllers on their hips and feet [14], which better constrains the space of possible poses, but comes at greater cost and inconvenience for the user. As of today, none of these solutions provide the ideal combination of fidelity, accuracy, and convenience.

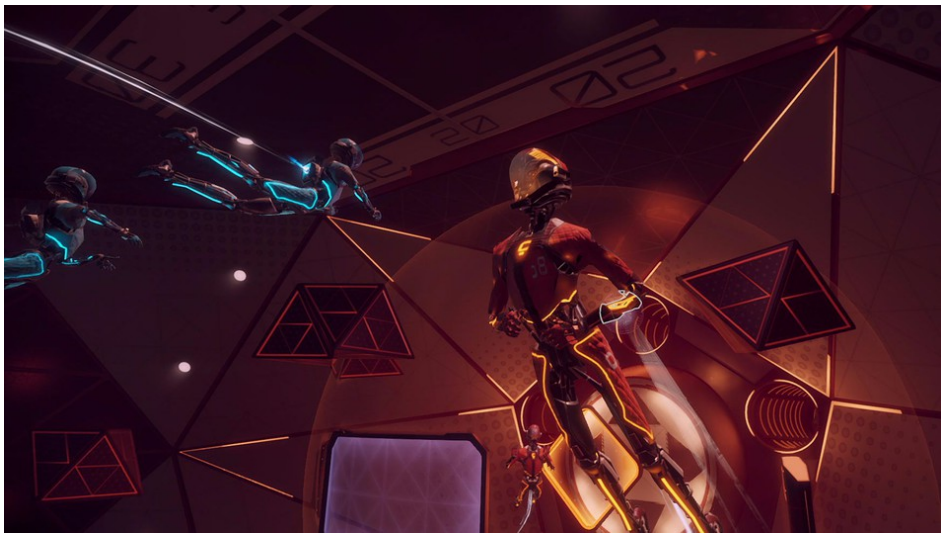

Figure 1.2: Screenshot from Echo Arena. Note the unnatural poses of the players in blue on the left. Picture taken from [2].

1.2 Recurrent Neural Networks

Recurrent neural networks (RNN) are a type of neural networks that specialize in dealing with temporal sequences. They store internal memory (often called "hidden state") that is considered when processing new input, and updated when outputting new predictions. RNNs and variants such as long short-term memory (LSTM) networks have been found to do well in use cases that are highly patterned over time, such as speech recognition or text completion.

RNNs are well-suited for tackling the problem of human pose estimation, because human motion itself is relatively well-determined when considering very small time-steps. Human joints cannot translate or rotate instantaneously, and typically continue along the path defined by its previous positions and rotations. By making use of temporal consistency, RNNs hold an advantage over other, history-blind algorithms when determining likely poses. Another advantage of RNNs (and data-driven methods in general) is in its model expressiveness: with sufficient data, they can potentially capture tendencies of human motion that traditional approaches such as IK cannot, resulting in more natural looking results.

1.3 Temporal IK

In this report, we introduce Temporal IK, a real-time, data-driven approach to full-body pose estimation for virtual reality applications. Given a stream of position and rotation values for the HMD and hand controllers over time, the model predicts corresponding joint angles and positions for a pre-defined skeletal structure. The underlying machine learning

model uses a RNN architecture with support for variable length sequences. It is trained with a newly captured dataset of an actor using VR applications, so as to better account for the style of movement typically seen when using virtual reality. The model is integrated into a package for the Oculus Rift VR system within the popular game engine Unity, so that it is more accessible for others to use in the future. We find that our solution does well in simulating natural movements of the user's upper body.

Contributions

Our contributions can be summarized as follows:

- We demonstrate an end-to-end solution for estimating full-body poses from the inputs provided by a standard VR system.
- We captured and make available 6.5 hours of motion capture data of actors using VR applications, constituting the largest publicly available dataset of VR motions and poses. We also provide simultaneously captured data of the positions and rotations of the HMD and controller during the motion capture sessions.
- We develop a package for parsing and visualizing the Biovision Hierarchy (BVH) motion capture data format within Unity.

Chapter 2

Related Work

There is a rich literature on the subject of human pose estimation, studied across a wide range of contexts, target use cases, and solution strategies. In the sections below, we provide background on the research that has been done in this space.

2.1 Input Modalities

Many methods for capturing data on human poses and motion exist. Motion capture systems use markers and extensive arrays of cameras to triangulate and fully determine the position of a person's joints at high frame rates. VR systems take a similar, albeit down-scaled version of this approach to track individual objects in the environment. In recent years, VR suits such as the Holosuit [20] or Teslasuit [8] have been developed with the goal of extending this tracking to the rest of the body. Unfortunately, both motion capture systems and VR suits are prohibitively expensive both in cost and setup time, and are unlikely to be accessible to most consumers anytime soon.

To improve accessibility, researchers have looked towards smarter solutions that reconstruct human poses using limited information from lower cost sensors. For instance, inertial measurement units (IMUs) capture information on their own orientation and linear/angular acceleration, and can be made quite cheaply. Several researchers have done studies where IMUs are attached to different points on a subject's body, and their readings are fused together to estimate the subject's movement [13][22][30][26]. The downside of this approach is that reasonably costed IMUs are quite noisy, and users still have to go through a significant setup process to wear and calibrate the sensors.

Computer vision based approaches have also been developed to take advantage of the power and convenience of consumer cameras. OpenPose [4] and Human Mesh Recovery [15] have shown promising results in predicting 2D and 3D human poses respectively from video footage. Private companies are also investing resources into developing solutions, with both Intel [19] and Facebook [12] recently showing off research in camera-based body tracking. A visual from the latter is shown in Figure 2.1

Figure 2.1: A researcher being tracked by Facebook's camera-based tracking solution [12].

Finally, some studies have utilized less orthodox methods of data collection. Microsoft's Kinect, initially marketed as a video game console, uses a depth sensor to perform human pose estimation [17]. Yin and Pai [34] use a foot pressure sensor pad to detect the movement of the feet and weight distribution of the body, and use that information to find motions that produce similar features.

2.2 Motion Mapping

Most research in the area of human pose estimation can be distilled to the problem of mapping an - often lower dimensional - feature vector to the correct motion or pose. Liu et al. [21] uses a combination of principal component analysis (PCA) and clustering to reduce a full set of tracked markers to a smaller dimensional one that still captures most of the variance in their dataset. They model human motion as a set of local linear models, and string together a piecewise sequence of these models based off of data from the reduced marker set.

Many other studies either build off of this idea or develop in a similar direction. Chai and Hodgins [5] create a data structure to support fast nearest-neighbor search of the set of local models. Safonova et al. [29] use PCA to find an initial guess, then use a physics simulator to adjust the pose to fit a set of given constraints. Kim et al. [16] also use an accelerated search method for find similar poses from a motion library, but use canonical correlation analysis (CCA) to build nonlinear local models to represent the motion instead. While these solutions do well at building a robust model relating the input features to motions, they end up with one or more disadvantages, such as being too slow, needing to store a large library of motion capture data, or producing unnatural movements.

2.3 Inverse Kinematics

Inverse kinematics is the process of determining the pose of a kinematic skeleton such that it satisfies a set of constraints. Often times, these constraints are used to set an end point to a particular position or guide it along a path. IK is commonly used in the field of robotics to plan and control the movement of robotic arms. It is also used in animation to adapt static animations to new environments. For example, characters with artist-defined running animations can use inverse kinematics to prevent the character's feet from sliding across the ground [18].

Inverse kinematics has also been applied to human pose estimation and animation, particularly in virtual reality. Given the positions of the body's "end-effectors", namely the head, hands, and feet, the limbs of the body can be treated as a robotic linkage moving back to some central joint, usually near the hip. Unzueta et al. [31] and Roth et al. [27] do exactly this, and the former uses a ball-and-socket joint limit model to avoid physically impossible poses. Meredith and Maddock [23] adapt an IK solver with a weight system to support a level of user-defined individualization, allowing for motion retargetting across differently shaped characters. Finally, Grochow et al. [9] develop a probabilistic model to explore the solution space of any IK problem. Depending on the data the model trains on, the IK solver will prefer solutions with a similar style of motion.

Figure 2.2: Screenshot from the game Lone Echo [3].

Several existing VR applications make use of IK to animate player avatars. For instance, the game Lone Echo by Ready at Dawn uses IK on both the avatar's arms and fingers [6], which allows the avatar to properly grab onto ledges and handles in the environment, as shown in Figure 2.2. The downside of inverse kinematic solutions is that they often result in unnatural movements. The imposed constraints on a kinematic skeleton leave the system

under-constrained, so multiple solutions exist. By themselves, IK solvers are unable to choose from these solutions in a manner that leads to natural or valid human poses.

2.4 Learning Approaches

In recent years, the popularization of machine learning has inspired new learning-based approaches to the problem of human pose estimation and animation. For example, the company DeepMotion uses deep reinforcement learning to develop physically-based models that keep an avatar standing and balanced while still fulfilling standard IK constraints [11]. Peng et al. [24] take a similar approach, but train their model to imitate pre-defined actions such as jumping or cartwheels. These studies show great promise in advancing the field of character animation, but don't optimize for replicating the pose of the user.

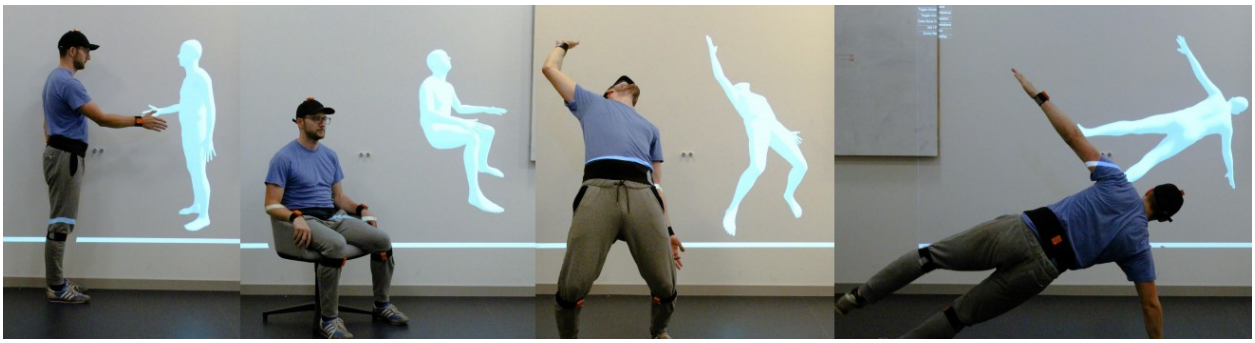

Figure 2.3: Pose predictions from Deep Inertial Poser [13].

In the domain of pose estimation, Huang et al. [13] introduces Deep Inertial Poser, a bi-directional RNN model that estimates human body poses based off of IMU readings. They synthesize a new dataset of IMU data and train their model to account for the noise unavoidable in affordable IMUs. Their paper show promising results, shown in Figure 2.3 using six IMUs attached to the user's body. In this report, we adopt some of their proposed strategies but operate in the input domain of VR tracked objects, which are fewer in number but provide more degrees of freedom at higher precision.

Chapter 3

Methods

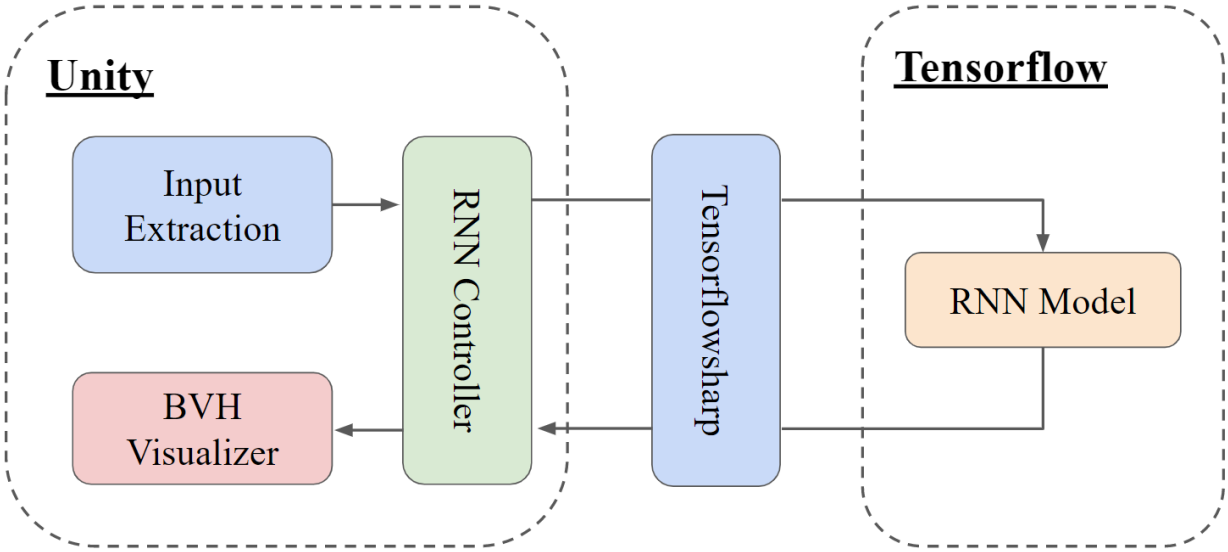

Figure 3.1: Overview of Temporal IK.

The Temporal IK workflow, as shown in Figure 3.1, operates across Unity and Tensorflow. While the end-to-end solution is built for Oculus Rift, Temporal IK can be easily applied to any VR system that supports 6-DOF tracking on the headset and controllers. The real-time process starts in the input extraction module, where the Oculus SDK is queried for the position and rotation of the headset and controllers. The data is sent to the RNN controller, where it is pre-processed and sent to the Tensorflow model via the Tensorflowsharp Unity plugin. The RNN model runs through the data and outputs human pose predictions, which are then passed along through Tensorflowsharp and the RNN controller until it is received by the BVH visualizer, where the predictions are displayed.

In the following sections, we provide an overview of the supporting work that was done to train and develop Temporal IK. We describe the motion capture process, BVH file format

and visualizer, data processing procedure, and RNN model architecture.

3.1 Motion Capture

To train our RNN model, we produce our own motion capture dataset and capture both standard motion capture data as well as the positions and rotations of the HMD and controllers. We choose to do this instead of adapting existing motion capture libraries due to our specialized target use case. In virtual reality, due to the constraints of the hardware, different types of motions are encouraged or prohibited. For example, users are unlikely to do cartwheels or run around while wearing a VR headset, but they might be encouraged to swing their sword or shoot a gun at a virtual enemy, which are motions typically not seen in real life. By training on data of people using VR applications, our model can better fit itself to the style of motions typically encountered in VR.

Physical Setup

Figure 3.2: Our data capture setup. One laptop (middle) runs and records VR inputs, while a second (left) operates the motion capture system.

Our data capture setup is shown in Fig 3.2, and takes place in a lab equipped with the Optitrack Prime 17W camera system. A table is set up at the front of the lab with an Oculus

Rift and laptop, setup to run standard Oculus apps and games, while a custom data collection process records the positions and rotations of tracked VR objects in the background into a txt file. A laptop on an adjacent side of the room runs Motive, Optitrack's software for camera operation, and exports motion capture data in the BVH file format. Our actor wears a motion capture suit and VR headset, as shown in Figure 3.3.

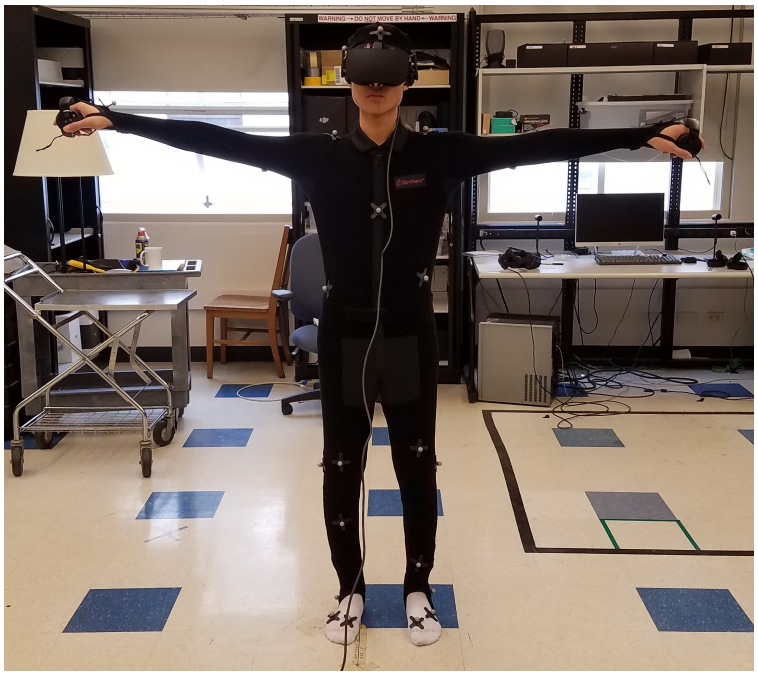

Figure 3.3: Our actor in a T-pose while in the motion capture suit and VR headset. Markers are placed according to the configuration of Optitrack's baseline markerset.

The recording process involves two people: an actor and a technician. After all necessary camera calibration and setup is completed, the procedure for starting a recording is as follows:

1. The actor stands in the center of the play space and holds the T-pose seen in Figure 3.3.

2. The technician moves to the motion capture laptop and calibrates the skeleton of the actor within Motive, thus setting the default pose of the motion capture avatar.

3. The technician moves to the VR laptop and calibrates the data collection process, which records the current attributes of the HMD and controllers as a calibration frame.

4. The technician begins recording in both systems.

5. The actor releases the T-pose, then jumps three times. This procedure helps determine the temporal offset between the two recordings later on.

Temporal Synchronization

For the captured VR input data to be useful, there needs to be a known temporal mapping between a recording of VR inputs and the corresponding motion capture data. Learning this mapping is a non-trivial task because both the coordinate system and origin of the two sensor systems are different. In addition, there are no known correspondence points (for example, markers on the head are offset from the HMD by an unknown amount).

To solve this problem, we note that the vertical y-axis for both sensor systems are the same in world space, or at least nearly identical to one another. So we take the y-coordinates from a VR tracked object in the VR input file, and extract the y-coordinates of the closest marker in the motion capture file. Doing so gives us two one-dimensional sequences to align.

We find the desired offset between these two sequences through cross-correlation. The two sequences are up-sampled to a common FPS so that only integer offsets need to be considered - in this case, the 120 FPS motion capture file and 50 FPS VR data file are up-sampled to 600 FPS. Then, for every possible offset within a reasonable window, the offset is processed and the dot product between the two sequences is computed to produce a correlation score. The offset with the highest such score is our predicted offset.

To verify this offset and make sure sequences don't have an issue of drift due to dropping frames or failing to meet their nominal FPS, we split the sequences into small chunks and repeat the procedure for each pair of chunks. This procedure allows us to confirm that the predicted offset continues to have the highest correlation score throughout the sequence.

3.2 BVH Visualization

In order to validate the data that we gathered as well as quickly visualize the predictions from the RNN model, it is necessary to have a way of parsing and visualizing motion capture data. To this end, we develop a Unity package that supports the parsing and visualization of the Biovision Hierarchy (BVH) file format, a plain text method of encoding motion capture data.

BVH File Format

The BVH file format is relatively well-documented [1], but we provide a brief description here. Each BVH file is split into two parts. In the first section, the skeletal joint hierarchy and calibration pose are defined. Joints are listed in the tree-like structure shown in Figure 3.4, where each one is described with a name, a local translational offset from its parent, and its channels of manipulation (such as rotation around the Z-axis or translation on the X-axis). The set of channels across all joints then fully defines the appearance of the skeleton.

In the second section, the motion of the skeleton over time is described. Every line of this section represents a single frame of time, and consists of a list of floats. Each float maps to one of the channels defined in the first section. Metadata such as the number of frames and time per frame are also provided.

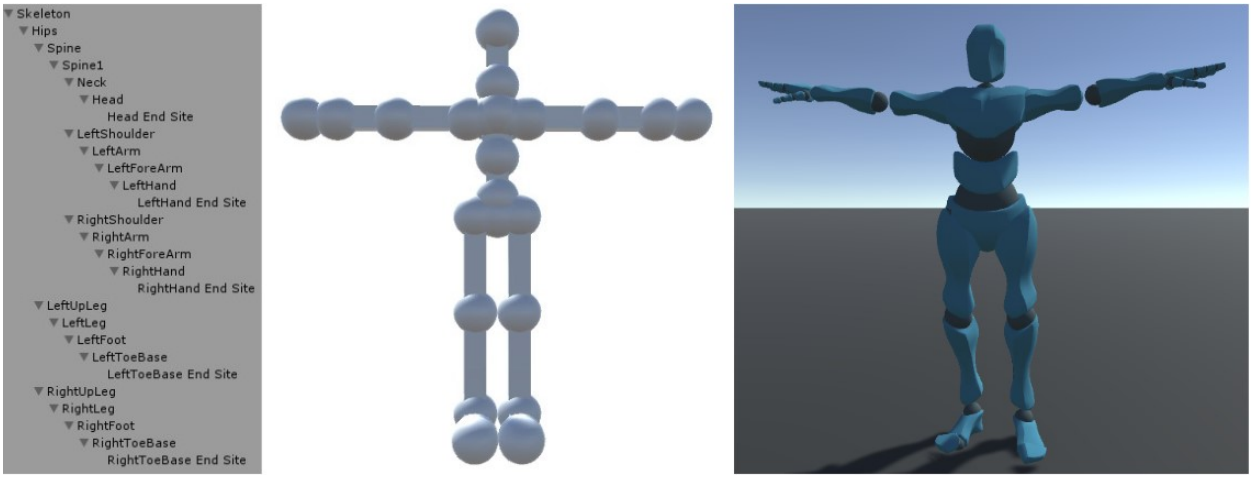

Figure 3.4: Left: joint hierarchy generated when exporting motion capture data. Center: visualization of a BVH file's calibration frame in Unity via primitive spheres and cubes. Right: controlling the appearance of an imported character model through the skeleton.

Visualization

Given a skeleton and frame of motion, we compute the resulting pose by computing the global position and rotation of each joint via forward kinematics. For any target joint n, we consider the chain of connected joints moving from it to joint $n-1$, $n-2$ etc. until it reaches the root joint 0. Within this chain, we can relate the global transformation matrix of a joint to that of its parent via the following equation:

$$X_n = R_n T_n X_{n-1}$$

Where X_n is the global transformation of joint n, T_n is a transformation representing the default local offset of joint n from its parent $n-1$, and R_n is the local transformation defined by the channels of joint n (in practice, this is always a rotation). We can then recursively apply this relationship down the chain until we get:

$$X_n = R_n T_n R_{n-1} T_{n-1} R_{n-2} T_{n-2} \dots X_0$$

This equation consists of local offsets, channel values, and the global transformation of the root joint, all of which are provided either in the skeletal definition or in the frame of motion. We compute this transformation for each joint to determine the structural appearance of the skeleton. For the graphical visualization, we start off by rendering spheres at each joint and connect adjacent joints with rectangular prisms. Later, we use the transformation data to drive the appearance of an imported character model. Example visualizations are shown in Figure 3.4.

3.3 Data Processing

Our aim is to train an RNN that, when fed data on the positions and rotations of the HMD and controllers, outputs the necessary attributes for all joints in a pre-defined skeleton. As shown in Figure 3.5, the input is a 18 element vector that concatenates the xyz position and xyz euler angle for the HMD, left controller, and right controller. The output is a 66 element vector that defines a motion frame in the BVH format.

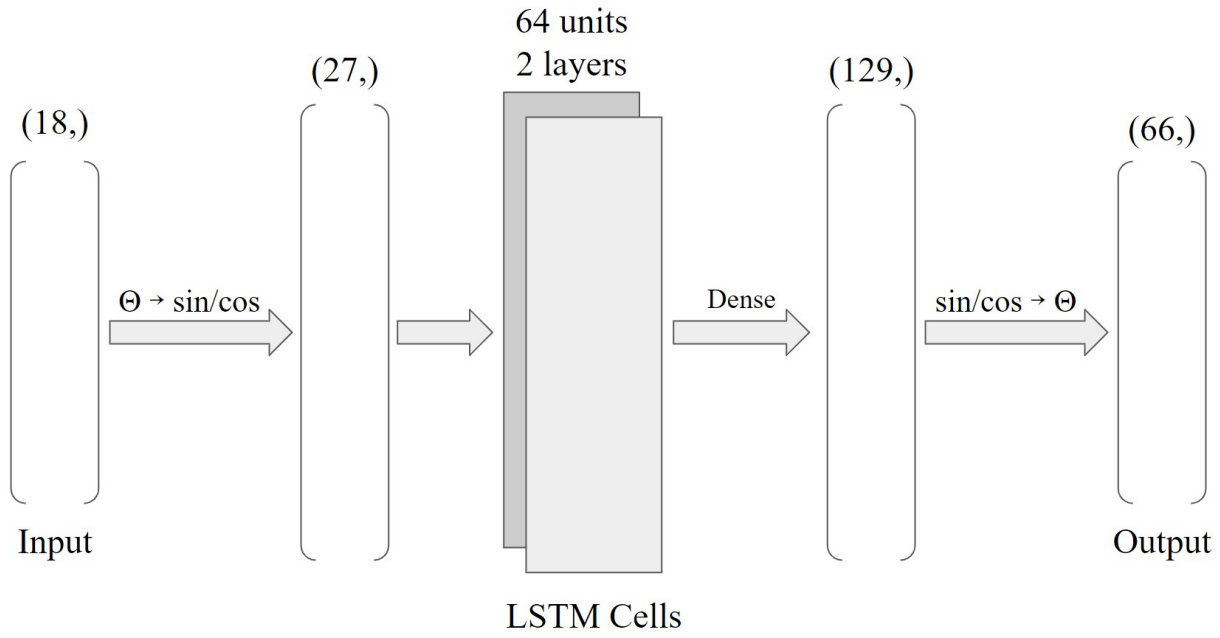

Figure 3.5: Architecture of our model.

Data Synthesis

We do not train our model using the raw recorded data from the VR system, opting instead to synthesize artificial inputs from the motion capture data. Using the forward kinematic equations described in the previous section, we extract the global positions and rotations of the skeletal joint closest to each tracked object and use those as the inputs when training our network.

We are able to make this substitution only by assuming that the substituted joint and corresponding VR tracked object are related to one another by a constant rigid-body transformation. If this is the case, we can determine and account for this transformation in Unity before feeding inputs into our RNN model. Since the HMD is fixed on the head and users will typically maintain a stable grip on the controllers, we deem this a safe assumption to make.

Synthesizing the inputs for our training data comes with two advantages. The first is that it prevents any error that would otherwise be caused by syncing up data between the two capture systems. The second is that there is no need to account for the change in coordinate system between the two capture systems, which differs between recordings.

Pre-processing

Before training our model, we perform a few pre-processing steps on our dataset. We start by downsampling our data from its original 120 FPS to 50 FPS to match Unity's internal clock speed. We use standard linear interpolation to do this, with some edge cases to deal with the periodicity of euler angles.

Next, we account for the coordinate system differences between Unity and the Motive software. Unity uses a left-handed, y-up coordinate system, where rotations are applied in the order of y, x, and z. Motive uses a right-handed, y-up coordinate system, where rotations are applied in the order of z, x, and y. We convert our motion capture data to Unity's coordinate system by negating all x-position, z-rotation, and y-rotation values.

Finally, motivated by prior experimentation [33], we replace each angle θ in the input and output vectors with a $sin(\theta)$, $cos(\theta)$ pair instead. The model has an easier time learning with this encoding, as it prevents the neural network from having to learn to replicate trigonometric functions.

Augmentation

To make our model invariant to the starting position of the user on the horizontal, or xz plane, we use an online augmentation approach: before training on a batch of data, we add a random translation to all x and z positions in the sequence of frames, essentially changing its starting point.

Our model is also made invariant to the starting direction the user is facing, i.e. rotation on the y-axis, through an offline approach. We note that the only values that need to be modified in the mocap capture dataset are the root joint attributes. So during the input data synthesis process, we repeat the process multiple times with random y-rotations applied to the root joint, and separately store the channel attributes of the root joint for each augmentation. Then during training, when we gather the labels for a batch of data, we update the root joint channels to match the y-rotation augment we're using.

3.4 Model Architecture

Our model architecture is shown in Figure 3.5, and is built and trained in Tensorflow 1.8. It consists of two layers of LSTM cells with an output and hidden state size of 64, followed by a fully-connected layer that maps to the dimensionality of the output. We train using the Adam Optimizer with a learning rate of 0.001, and add a dropout chance of 0.2 to the

weights to prevent overfitting. We train on sequences of length 32 (0.64 seconds at 50 FPS), and use 512 sequences per batch. The model is trained on roughly 5 hours of captured data, and takes about 2 hours on a GTX 1060 to run to asymptotic error.

Chapter 4

Results

To evaluate Temporal IK, we run it on test sequences of synthesized input data and compare the results to the corresponding motion capture data. We pick out two sequences, detailed in Table 4.1. The first sequence is representative of the style of movement that we trained our model on. The second sequence contains more artificial poses and movements not present in our dataset.

Sequence	Duration	Actions
1	2:47	Beat saber (rhythm game), general movements
2	4:55	Static poses, jumps, squats, menu manipulation

Table 4.1: Meta information for the two test sequences.

In the sections below, we discuss the results of our experiments, along with insights derived from them.

4.1 Quantitative

We quantitatively evaluate Temporal IK through two error metrics: mean skeletal error and mean distance from target positions. To compute the former, we consider each of the joints in the BVH skeleton separately. For each one, we take the absolute deviation between the predicted rotation and what's defined in the motion capture data, and average that over time. We separately consider the average deviation of the root joint's position as well.

The results can be found in Table 4.2. As expected, the error jumps up for the test sequence with motions less familiar to the model. The increase in error is seen mostly around the arm joints, which can likely be attributed to the fact that the style of movement in a VR experience is defined mostly by how the arms move. Interestingly, the error for the leg joints aren't significantly higher than that of the arm joints, despite there being less information about them in the input data. We discuss why this happens in the qualitative section.

Sequence	**Joint**			
	Spine	**Spine1**	**Neck**	**Head**
1	2.30	1.74	3.57	4.04
2	2.78	2.40	4.15	5.53
	LeftShoulder	**LeftArm**	**LeftForeArm**	**LeftHand**
1	3.83	5.01	5.04	4.69
2	3.20	8.75	5.50	4.26
	RightShoulder	**RightArm**	**RightForeArm**	**RightHand**
1	3.28	5.23	6.02	4.09
2	3.22	7.70	10.76	6.90
	LeftUpLeg	**LeftLeg**	**LeftRoot**	**LeftToeBase**
1	3.09	3.51	5.36	0.00
2	4.61	3.97	5.68	0.00
	RightUpLeg	**RightLeg**	**RightFoot**	**RightToeBase**
1	3.33	3.05	4.14	0.00
2	4.30	3.54	5.18	0.00
	Hip (pos)	**Hip (rot)**	**Average (rot)**	
1	2.49	2.93	3.54	
2	3.67	6.59	4.72	

Table 4.2: Mean absolute error (MAE) for different joints in the BVH skeleton. Error is averaged across all frames for the two motion sequences described in Table 4.1. Errors are in degrees, except for Hip (pos) which is in cm. Average error does not include Hip (pos).

As discussed early on in the introduction, VR avatars have to be responsive to movements of the user. Specifically, the virtual head hands produced by this model must closely mimic the movement of the user's controllers in real life. To test the performance of our model on this point, we compute the average distance between each tracked input object and its associated skeletal joint over the duration of the sequence.

The results are shown in Table 4.3. Just as with the skeletal error, average error is lower for the more familiar test sequence.

Sequences	**Head**	**Left Hand**	**Right Hand**	**Average**
1	2.05	2.30	2.28	2.21
2	3.26	2.84	3.31	3.13

Table 4.3: Average distance (cm) between tracked input objects and their corresponding skeletal joints. Error averaged across all samples for two motion sequences, described in Table 4.1.

Finally, we profile a normal Unity scene running Temporal IK, where we set the model to be evaluated and output predictions every 0.05s (20 FPS). We find that the scene runs

a little over 20 FPS. This number goes down to roughly 10 FPS when we run the scene in VR, as VR adds a significant computational burden to the application. It is worth noting that these values are not optimized, and could be much higher on a stronger computer or with an optimized build of Temporal IK.

4.2 Qualitative

Temporal IK is ultimately a visuals-based project, and so to determine how our model is behaving we look towards qualitative analyses. To start off, we run Temporal IK on another test sequence and construct a side-by-side comparison between the model's live predictions and the original motion capture data. We display a few frames of this comparison in Figure 4.2.

Temporal IK does well at reconstructing the movement of the arms. The overall motions are correct, and the model generally avoids poses or motions that are impossible to replicate in real life. Certain quirks of human motion are picked up: for instance, when the user makes a horizontal swiping motion, the model will lead with its elbow first.

One hope for Temporal IK was that the model would learn to position feet at or above the ground plane. Unfortunately, the results do not show this. Instead, the lower body is kept at a mostly static pose that represents the "average" position of the lower body. When the user jumps, walks around, or does other leg movements, the model usually fails to imitate the action. If the height of the head decreases significantly, the model adopts a squatting pose. Examples of both of these situations are shown in Figure 4.1. In our training data, the subject mostly stands upright and rarely moves their legs, which is why the quantitative error for lower body joints isn't very high compared to the upper body.

Another note about the model's behavior is that it lacks "weight". Movements are occasionally jittery, and the center of the gravity fluctuates, causing the avatar to feel not as grounded as the motion capture data.

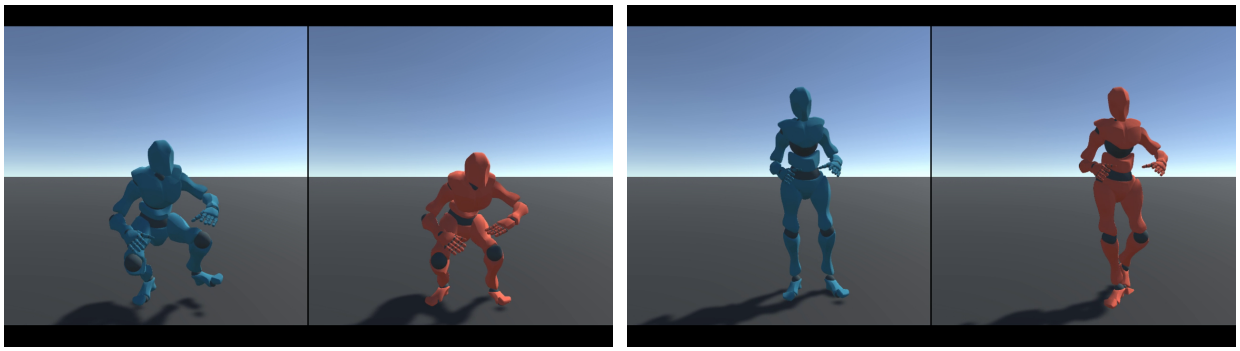

Figure 4.1: Left: the model (blue robot on the left) squats in response to the user ducking their head. Right: the user physically takes a step, but the model keeps the legs static.

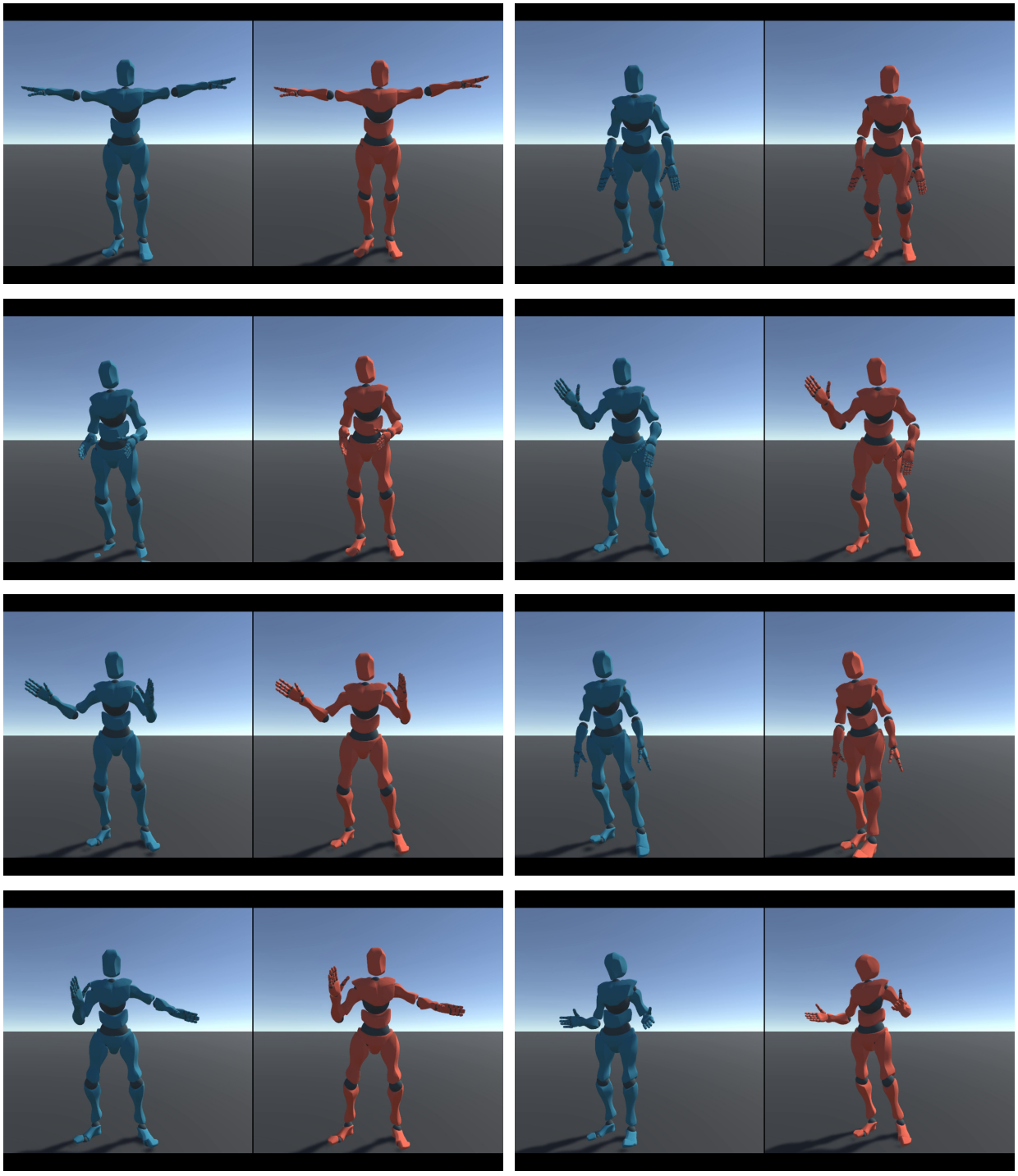

Figure 4.2: A side by side comparison on a test sequence. In all images, red on the right is the original motion capture data, while blue on the left is the live prediction made by Temporal IK.

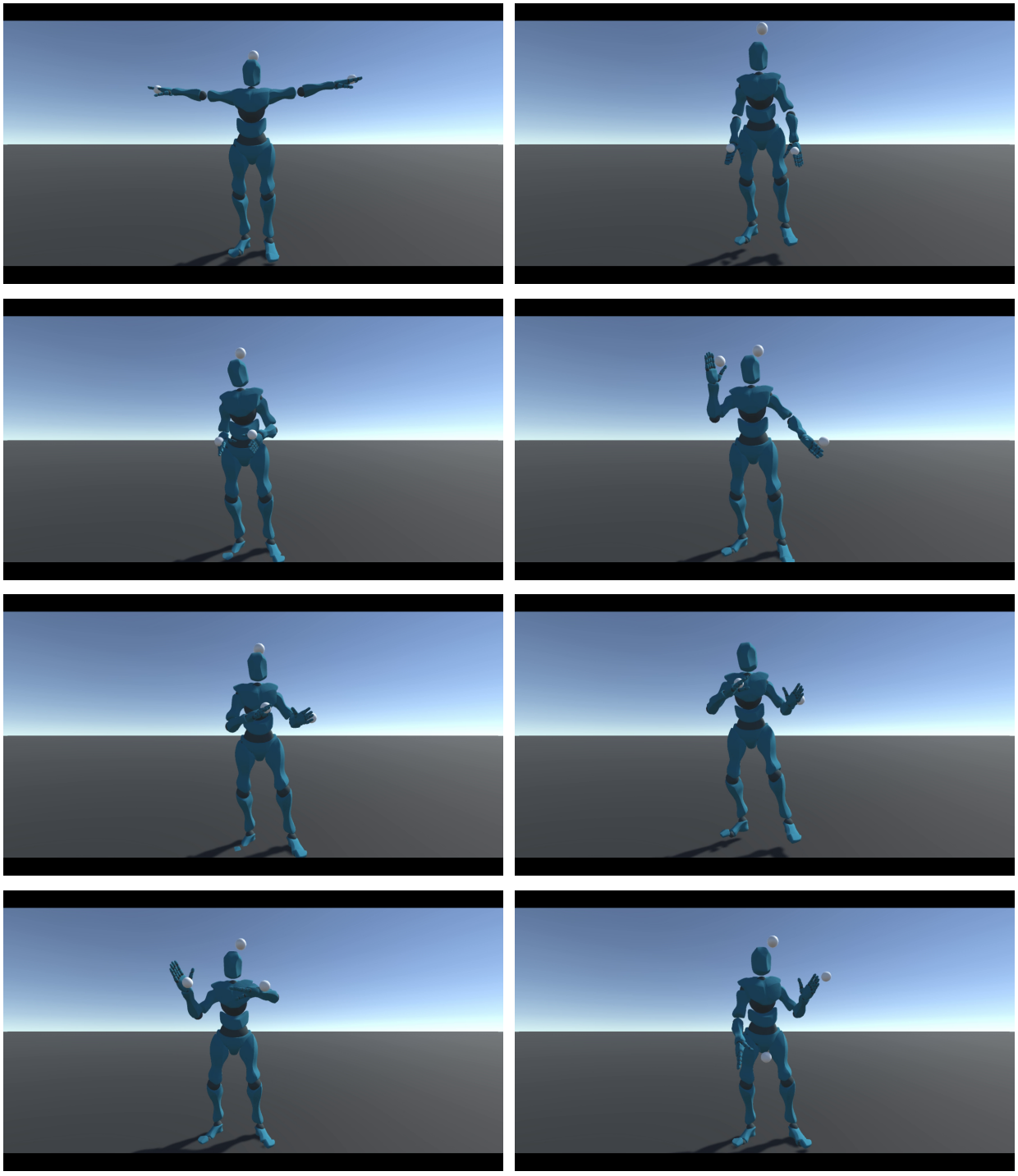

Figure 4.3: The model evaluated on a test sequence, overlaid with the data that was fed into the model (i.e. the VR inputs). The white spheres represent tracked objects, and map to the two hands and top of the head.

Chapter 5

Conclusion

5.1 Summary

In this report, we introduce Temporal IK, a data-driven approach to solving the problem of human pose estimation in the context of virtual reality applications. We develop and train a recurrent neural network that predicts motions from sequences of positions and rotations from a user's VR headset and controllers, seeking to exploit the temporal consistency exhibited by most human movements. As part of the work towards this goal, we record several hours of motion capture footage of an actor using virtual reality, accompanied by corresponding data from the VR tracked objects. In addition, we develop a real-time Unity integration package that supports both Temporal IK and general motion capture data parsing/visualization.

We observe that our solution does well for the upper body, and produces natural looking movements that imitate the true motion closely. However, a fundamental lack of input data associated with the lower half of the body makes it difficult for the model to accurately predict the movement of the legs.

5.2 Future Work

There are a number of directions that Temporal IK could be taken in to push this topic of research further. For example, the outputs of the model are currently being applied raw, and post-processing could be applied to improve the visualizations. Temporal smoothing could be applied to reduce the jitter and flucations noticeable in the current system, and inverse kinematics could be used on the arms and head to clamp down any discrepancies between the target position and the RNN's prediction.

Regarding the neural network itself: transfer learning techniques, which have grown in popularity the past few years, could be used to improve the training process of Temporal IK. A pre-training dataset consisting of standard motions, perhaps including pre-existing motion capture datasets, could be applied first. Then, data from a specific application could be used

to fine-tune the network, thus specializing the model to that application's style of motion. It would also be interesting to explore and compare other machine learning architectures that take advantage of temporal patterns, such as Google's WaveNet [32]. In addition, different methods of representing error could be tried - for example, a future model might include error terms that represent the distance between the head and hand joints and their respective input controllers so as to put greater emphasis on those joints.

Improving Temporal IK's ability to convincingly visualize the lower body is critical. One possible way would be to learn when the user's feet are planted on the ground, then enforce that constraint either in the model itself or in post-processing through inverse kinematics. Alternatively, a hybrid approach could be taken, where the upper body is controlled by the RNN but the lower body is controlled through some other mechanism, such as DeepMotion's physics based controller.

Bibliography

[1] Jeff Ballard. *Biovision BVH*. Feb. 1999. URL: https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html.

[2] Kellen Beck. *'Echo Arena' is the first VR game that made me forget I was real*. June 2017. URL: https://mashable.com/2017/06/21/echo-arena-vr/.

[3] Dominic Brennan. *'Lone Echo' Developer Shows Impressive Procedural Hand-posing System for VR - Road to VR*. 2017. URL: https://www.roadtovr.com/lone-echo-developer-shows-impressive-procedural-hand-posing-system-vr/.

[4] Zhe Cao et al. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields". In: *arXiv preprint arXiv:1812.08008* (2018).

[5] Jinxiang Chai and Jessica K Hodgins. "Performance animation from low-dimensional control signals". In: *ACM Transactions on Graphics (ToG)* 24.3 (2005), pp. 686–696.

[6] Jacob Copenhaver. *VR Animation and Locomotion Systems in Lone Echo*. 2017. URL: https://readyatdawn.sharefile.com/share/view/s4565f1ec6e046469.

[7] Ready at Dawn. *Echo Arena*. [digital]. 2017.

[8] Susan Fourtane. *Teslasuit Brings Virtual Reality to a New Level: Not Just for Gaming*. Sept. 2018. URL: https://interestingengineering.com/teslasuit-brings-virtual-reality-to-a-new-level-not-just-for-gaming.

[9] Keith Grochow et al. "Style-based inverse kinematics". In: *ACM transactions on graphics (TOG)*. Vol. 23. 3. ACM. 2004, pp. 522–531.

[10] Scott Hayden. *Magic Leap's AI 'Mica' Won't Turn Off Your Lights, Play Your Music, or Give Directions*. Jan. 2019. URL: https://www.roadtovr.com/magic-leaps-ai-mica-wont-turn-off-lights-play-music-give-directions/.

[11] Kevin He. *Using Deep Learning to Create Interactive Actors for VR*. DeepMotion. 2018. URL: https://youtu.be/Gx_Twp943qE.

[12] David Heaney. *Facebook Shows Off High Quality Realtime Body Tracking*. May 2019. URL: https://uploadvr.com/facebook-f8-2019-body-tracking/.

[13] Yinghao Huang et al. "Deep inertial poser: learning to reconstruct human pose from sparse inertial measurements in real time". In: *SIGGRAPH Asia 2018 Technical Papers*. ACM. 2018, p. 185.

[14] VRChat Inc. *VRChat*. [digital]. 2017.

[15] Angjoo Kanazawa et al. "End-to-end recovery of human shape and pose". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7122–7131.

[16] Jongmin Kim, Yeongho Seol, and Jehee Lee. "Human motion reconstruction from sparse 3D motion sensors using kernel CCA-based regression". In: *Computer Animation and Virtual Worlds* 24.6 (2013), pp. 565–576.

[17] Pushmeet Kohli and Jamie Shotton. *Key Developments in Human Pose Estimation for Kinect*. Consumer Depth Cameras for Computer Vision. Springer, Jan. 2013. URL: https://www.microsoft.com/en-us/research/publication/key-developments-in-human-pose-estimation-for-kinect/.

[18] Lucas Kovar, John Schreiner, and Michael Gleicher. "Footskate cleanup for motion capture editing". In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM. 2002, pp. 97–104.

[19] Philip Krejov. *Deep Learning for VR/AR: Body tracking with Intel RealSense Technology*. Intel. 2018. URL: https://youtu.be/VSHDyUXSNqY.

[20] Juanita Leatham. *HoloSuit Kickstarter to Fund Production of Smart Clothing for VR Fitness, Sports and Games*. June 2018. URL: https://www.vrfitnessinsider.com/holosuit-kickstarter-to-fund-production-of-smart-clothing-for-vr-fitness-sports-games/.

[21] Guodong Liu et al. "Human motion estimation from a reduced marker set". In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. ACM. 2006, pp. 35–42.

[22] Huajun Liu et al. "Realtime human motion control with a small number of inertial sensors". In: *Symposium on interactive 3D graphics and games*. ACM. 2011, pp. 133–140.

[23] Michael Meredith and Steve Maddock. "Adapting motion capture data using weighted real-time inverse kinematics". In: *Computers in Entertainment (CIE)* 3.1 (2005), pp. 5–5.

[24] Xue Bin Peng et al. "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), p. 143.

[25] Emily Reynolds. *Virtual Reality Makes Avatars More Important Than Ever*. Dec. 2016. URL: https://www.vice.com/en_us/article/pgkv9z/vr-makes-avatars-more-important-than-ever.

[26] Qaiser Riaz et al. "Motion reconstruction using very few accelerometers and ground contacts". In: *Graphical Models* 79 (2015), pp. 23–38.

[27] Daniel Roth et al. "A simplified inverse kinematic approach for embodied vr applications". In: *2016 IEEE Virtual Reality (VR)*. IEEE. 2016, pp. 275–276.

[28] Peter Rubin. *Facebook Can Make VR Avatars Look-and Move-Exactly Like You*. Mar. 2019. URL: https://www.wired.com/story/facebook-oculus-codec-avatars-vr/.

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

[30] Ronit Slyper and Jessica K Hodgins. "Action capture with accelerometers". In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association. 2008, pp. 193–199.

[31] Luis Unzueta et al. "Full-body performance animation with sequential inverse kinematics". In: *Graphical models* 70.5 (2008), pp. 87–104.

[32] Aaron Van Den Oord et al. "WaveNet: A generative model for raw audio." In: *SSW* 125 (2016).

[33] Lyndon White. *Encoding Angle Data for Neural Network*. 2016. URL: https://stats.stackexchange.com/q/218547.

[34] KangKang Yin and Dinesh K Pai. "Footsee: an interactive animation system". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2003, pp. 329–338.