Universiteit Leiden
The Netherlands

Computer Science & Advanced Data Analytics

Evaluating and Quantifying Biped Locomotion Naturalness in neural nets trained with Kinect motion data

Ruben van der Waal BSc

Supervisors:
Dr. Michael S. Lew
Dr. Erwin M. Bakker

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

27/08/2022

Abstract

Naturalness is an inherently human concept and a highly desirable quality for locomotion. When training any deep neural net approach to this we often want to try many different configurations. Simple Mean Squared Error and other out-of-the-box metrics do not paint the entire picture. In this thesis we study the validity of our own custom metrics. We correlate these metrics with human evaluation scores to get an insight on their effectiveness and what pertains to a good score. Our data comes from a low-cost setup of two Microsoft Kinect v2's. The actors are observed from two perspectives simultaneously to enhance accuracy. We have a good indication that our metrics are working as intended, however the high reliance on human evaluation led to a small amount of samples.

Acknowledgements

Firstly I would like to express my deepest gratitude to Dr. Michael S. Lew who provided advice, help, the hardware and mental clarity when I needed it. Without him this thesis would not have been possible.

Secondly I would like to thank my family, friends and girlfriend for their loving support throughout the time needed to complete this thesis.

Lastly I thank all the people that have helped me evaluate the work and results.

Contents

Chapter 1

Introduction

Humans have an instinctual ability to discern the natural from the unnatural. Anything that does not quite reach the acceptable level of naturalness falls in what is known as the uncanny valley. We aim to quantify the naturalness of a movement in an objective manner with the help of a human test group.

Generating locomotion is a widely researched and challenging topic. Various methods exist. A trio of papers come out in 2002 detailing motion graphs [KGP02] [AF02] [LCR+02]. Which is an elegant solution to calculate motion when a clear target pose and possible constraints along the way are specified. Finite state machines break down the movement into a set of actions and states [YLP07] [KL21]. On top of that there are the methods that use neural networks. Previously we did preliminary research [vdW20], which was based off of the work of Holden et al [HKS17] that used such networks. Within the neural net approach there is an emphasis on deep reinforcement learning as detailed in the works of [DCH+16] [YTL18]. Some of the aforementioned works are interested in simply moving the avatar in a physics-simulated environment and others in producing lifelike motion. In this work we focus on automatically evaluating the latter.

When going through the process of hyperparameter optimization we often have a large search space. Each combination of parameters produces its own net. A simple Mean Squared Error loss on a validation set does not paint the entire picture. Our work here is aimed at automatically assessing the perceived naturalness of the locomotion the net produces by applying our custom metrics. To achieve this we record a varied set of movements. For each movement we find an optimized recurrent net with an abundance of nodes via a series of grid searches. Each net generates animation by feeding the prediction back as the latest timestep of the input. The animation is the evaluated by a group of people to assess its naturalness in various ways. These scores are then correlated to our custom metrics. We can then use our custom metrics to directly reject or accept different combinations of parameters during the subsequent training of any locomotion generating net.

We first discuss some of the related work. Then describe our inexpensive recording setup using two Microsoft Kinect V2 and some of the issues we encountered. Then we show the process of selecting our movements and how the data is represented. We explain our custom metrics and our search strategy for finding a good fit of the hyperparameters. After that we show how the metrics performed on our data and how our questionnaire

participants evaluated the results. Lastly we show how our metrics and the human scoring correlate and discuss the results.

1.1 Contributions

We introduce a novel method of combining two avatars from different perspectives. Our own dataset contains five movements varied in upper and lower body activity. We outline the limitations of the Kinect when recording. For our network hyperparameter optimization we use a 3-stage grid search approach. In this case this means we segment the total search space to limit the amount of combinations. We propose our own novel widely applicable metrics to quantify the naturalness of a given animation. Our metrics are the Limb Length Error (LLE), Body Matching Error (BME) and Foot Slide Error (FSE). We compare the metrics to the results of a human study group.

Chapter 2

Related Literature

2.1 Previous Work

This work is an extension of our previous work [vdW20]. In which we focused on testing the limitations of the Kinect in regards to noise resistance and skeleton recognition accuracy, using a set of different movements. We employed a phase functioned neural network to learn our data, as seen in the work of Holden et al [HKS17]. We transferred the net to a C# implementation for online use in the Unity3D engine. It became apparent during the project that the Kinect itself was the limiting factor. The area in which actors could be effectively recorded was small. Many body positions are hard or impossible to recognize for the Kinect. If the actor faced away from the Kinect the tracking would also fail. We theorized that introducing a second Kinect could improve the recording. There are several options, we could:

a) Extend the observed area, by positioning the cameras side by side so that when the actor leaves one view it enters the other.

b) Increase the accuracy on a limited area, by positioning the cameras side by side with an overlapping view. The actor is now always observed by two cameras increasing the accuracy in case one Kinect fails.

c) Increase the options in a limited area, by positioning the cameras facing each other so that when the actor turns away from one camera it is now facing the other.

2.2 Stereo Vision

Stereo Vision in order to increase accuracy for computer vision tasks is by no means a new concept. In the work of Prasad et al [PSJ+11] we see a low-cost, effective setup to recognize gestures using two fixed webcams. There have also been solutions using the Kinect on gesture recognition as shown by Yi Li [Li12], Jiang et al [JZW+15] and Ren et al [RMYZ11]. Another work by Ulges et al [ULB04] shows the use of stereo vision to reconstruct the curve of a page of text in 3d and dewarp the text displayed. Without the second perspective only a simple perspective transformation would be realistic. Stereo vision using RGB cameras can even enhance the use of

3D LIDAR as seen in Gupta et al [GUGK17]. The false positives as well as false negatives in the detection of small obstacles on the road was decreased. It seems there's ample evidence to suggest a second perspective aids our work.

2.3 Multi-Kinect Setups

Utilizing more than one Kinect at once can cause difficulties due to the active light nature of the sensor. In Berger et al [BRS+11] the level of interference caused by this when capturing motion was studied. They concluded that the effects of interference between the Kinects are minimal when the Kinects are placed with a maximal viewing angle between them. The process of finding a common coordinate system was explored by Stewart et al [SMK19]. All three aforementioned options *abc* of incorporating multiple Kinects were explored. The setup used was a client server configuration where each Kinect had its own client which then send the data to a server that combined the data in a common coordinate system. This was done using various algorithms reliant on the Mean Squared Error between each view of each avatar. The setup was showcased with a project named Liminal Space [MJL18], which transforms dance movements into music. Asteriadis et al [ACZ+13] shows an application of multiple Kinects to see around occlusions. They show a setup in which an actor running on a treadmill who is partially occluded by the bars and dashboard could still be accurately reconstructed using the multiple perspectives. Ghose et al [GCAA13] introduced another type of setup that would fall under our option *a*. Kinects are placed throughout the subject's home and can identify an individual by their walking patterns and can track their activities during the day and night.

At the time of writing this paper a newer version of the Kinect exists named the Kinect Azure. This version of the Kinect natively supports multiple devices working together. It also recognizes a much wider variety of body poses. This opens up many new applications. In the works of Liu et al [LYLL22] the movement studied was a push up. The front of the body is entirely occluded in this scenario.

2.4 Training nets for generating locomotion data

Our previous work was based of the work of Holden et al [HSKJ15] in which a phase functioned neural network (PFNN) with a single hidden layer was employed. The PFNN shifts its weights according to the phase the movement is in. The weights are obtained by a function dependent on the phase. The function used here is a cubic Catmull-Rom spline with 4 control points. In this project we opted for a Recurrent Neural net approach such as the encoder-recurrent-decoder (ERD) proposed by Fragkiadaki et al [FLM15]. Although in this project we do not use the encoder-decoder setup. Yu et al [YTL18] shows a Deep reinforcement learning (DRL) approach to the problem of generating motion. It is difficult for DRL models to produce natural motion. They added a term to the loss function to encourage symmetric actions and the application of Curriculum Learning [BLCW09] which yielded good results.

2.5 Quantifying naturalness

Naturalness is a very human concept and remains a challenging research topic in many applications. In the work of Arima et al [AHK18] a probabilistic language model is employed to evaluate the naturalness of code. They compare inlining a method to extracting it and test it on a piece of professional code.
In Ramm et al [RGRM14] the idea of introducing a statistical model is not the goal for evaluating naturalness in Driver-Car interaction. Instead extensive interviews with drivers which lead to 10 main points of what defines the naturalness. The authors propose introducing a checklist for future evaluation of the Driver-Car interaction. This means human evaluation remains a necessity.
Nilson et al [NSN13] explores naturalness in immersive virtual reality environments. They compare different methods of Walking-In-Place by their perceived naturalness, presence and real world positional drift. They use a mix of human evaluation and objective measures.

Ren et al [RPE+05] has tried many statistical models such as, Mixture of Gaussians, hidden Markov Models and switching linear dynamic systems. They do this on a large database of roughly 4 hours of high grade motion capture data. They compare their findings to the results of a human evaluation. The data was largely skewed towards positive examples of natural motion. A problem they encountered was that the statistical approaches would only be good at recognizing errors that they had seen before. Unnaturalness of a sort not encapsulated in the data might not be labeled correctly.

Chapter 3

Definitions

- **Actor**: The real world person acting out the movements in front of the Kinects.
- **Avatar**: The virtual representation of the actor as detected by the Kinect software. Also known as skeleton.
- **Animation**: The avatar playing the movements as generated by the neural net.
- **Relative Joint Position**: The 3d world coordinates in a local coordinate system. The root joint of the avatar is the origin of the coordinate system. The XYZ axi line up with the XYZ axis of the the avatar
- **Absolute Joint Position** The 3d world coordinates of the joints of the avatar.
- **Joint Orientations** The unit quaternion that represents the orientation of each bone. Also referred to as joint rotations.
- **Movement** A specific sequence of actions or steps performed by the actor.

Chapter 4

Recording

4.1 Recording Setup

For our experiments we use two Microsoft Kinect V2. Multiple setups are possible. We could extend the observed area by having a minimal overlap between the camera views. Or we could maximize the overlap and have the focus on the overlapping area. With either option we should be able to get higher quality recordings since the actor is observed from different angles at the same time.

In this experiment we chose to maximally overlap the Kinects. The Kinects were positioned about 60 centimeters apart and about 110 centimeters high. The cameras were both aimed inwards slightly so their principal axes intersect at about 230 cm. This intersection is also the optimal point for the actor to perform.

We record from both Kinects at the same time. For each frame that arrives we note the time it is added to the processing queue. This is important since we can not process all the data as fast as it comes in. The timestamps are later used to synchronize the data streams. On each frame the NiTE2 framework also extracts the floor plane. The accuracy of this varied and we opted to estimate this ourselves based on the joint positions and visual inspection.
We also receive joint orientation data, which are represented as quaternions. However in our testing the rotations showed to be unstable. We determined that this is partly because joints and limbs are covered by clothing. The Kinect also calculates the rotation of the arm inversely from the hand position. Which can result in errors stacking up.
For both the joint positions and rotations we receive a confidence value. Upon inspection the variance in the confidence was low. Even in the event of catastrophic misdetection of the avatar the confidence would not dip below 0.6. Early efforts were made to generate a bias between the data streams based upon this quality measure. The position and rotation confidences were not correlated well enough with actual performance. For this reason we discarded the joint orientations. For each avatar we do still save the joint positions, floor plane and averaged confidence.

4.2 Avatar Detection

Now that we have our hardware and recording parameters we can start recording some sample motion. This immediately showed some difficulties with the hardware and the drivers:

- The Kinects do not always detect the actor. A workaround is to cover the Kinect's sensors momentarily. This seems to reset the detection algorithm and the actor is detected again. Sometimes this causes the other Kinect to lose tracking again.
- In some cases the alignment of the avatar with the actor in the depth view of the Kinect is off. Resetting the system fixes this sometimes. It can only be assumed the detection algorithm has found a bad source point from where it builds the avatar and does not self-correct. In example can be seen in Figure 4.1.
- When joints like the knee or elbow are bent the bone coming off this joint sometimes is mapped onto the bone before it. For example if the knee is bent the shin will be confused with the femur. The knee is then detected to be fully bent with the shin and the femur in the same place. This puts the foot near the hip. This is big error in positioning and this also affects our naturalness metrics and custom floor detection algorithm. Even if the avatar is averaged with the other perspective the error is too large and will affect the final recording. We discard the recording in this case.
- Similar to the previous difficulty, bones can also overlap with other bones. This most commonly happens with the legs being switched or a bent arm being detected as a straight arm if it is close to the torso. These errors are also non recoverable and the recording is rejected.
- The Kinect detects the avatar independent of previous frames. This is to avoid building off any mistakes in detection earlier. The downside of this is that joint tracking can be unstable. Which results in joints detected by the Kinect moving even with the actor standing still. We counter this by averaging the joint positions with their previous and next position.
- There is a minimum distance from the actor to the Kinect to be able to detect properly. If we do not comply to this we cannot properly record the actor and the floor plane detection fails.
- Avatar detection from the side or back is not be supported. Body detection degrades significantly after the shoulders and hips face away more than 30 degrees and by the point it reaches 45 degrees the avatar is unusable.
- Another issue is the large amount of data transferred. The USB transfer pool needs to be sufficiently large to accommodate data from both Kinects coming in.

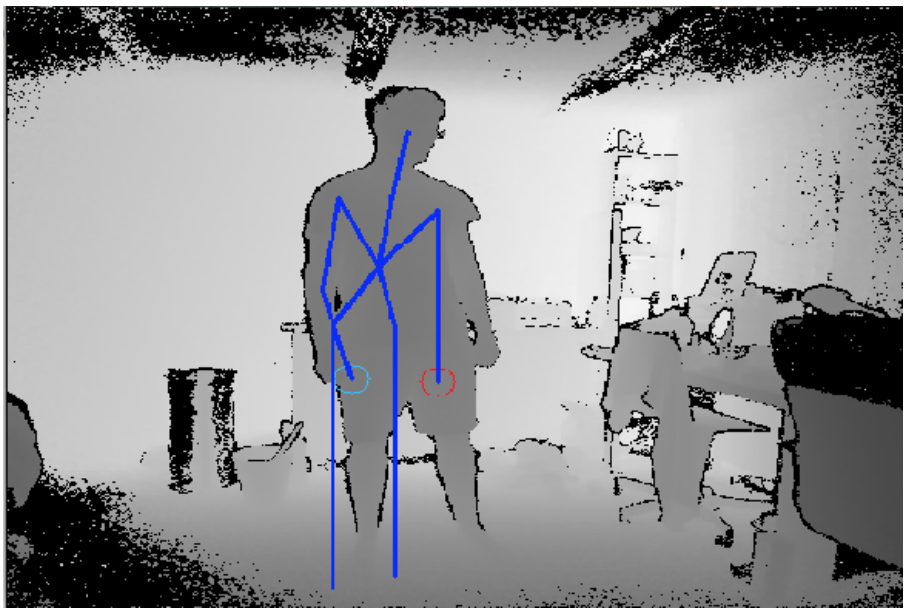


Figure 4.1: Example of misalignment

These difficulties put immediate restrictions on what type of movement can be recorded. We cannot have the actor spin or hold their arms too much in front of the torso. Also the area in which they can move around is limited, since both camera's need to be able to observe the actor. However the speed of the movement does not seem to impact the quality of the tracking.

Chapter 5

Homography

Our dual Kinect setup records the body from two different camera perspectives, with different coordinate systems. We need an accurate transformation matrix to link the two perspectives.

5.1 Chessboard-based Homography

Most approaches of finding a transformation require a set of matching 3d points in both camera coordinate systems. For any of those methods to work well we first need accurate camera matrices and distortion coefficients. The Kinect comes with factory specifications that can be read from the device. However the camera matrices and distortion coefficients are not accurate enough and we will start with determining these ourselves. To do this we take an easily recognizable object with known real world dimensions. We used a printed chessboard pattern. Though we refer to this as the chessboard pattern it is not actually an eight by eight checker pattern, it is ten by seven pattern. The asymmetrical shape combined with the black and white pattern means we can always detect the corners in the same order. The pattern is held up in front of the camera. It is important for good calibration to hold the pattern at different angles and distances and also throughout the field of view of the camera. We take 50 sample images. All corners of the chessboard create a vast amount of data points. We refer to the corner positions in the image as image points. For each 2d image point we construct a matching 3d object point. The 3d object points are the real world dimensions of the chessboard at distance 0. We then find the optimal camera matrix and distortion coefficients that map all the 3d object points to the corresponding 2d image points. To speed up this process we can start with the camera matrix given by the manufacturer.

Now that we have our more accurate camera matrix we can focus on transforming between the cameras. As a proof of concept we first attempt to transform between the RGB camera and IR camera on the same Kinect. These cameras are only a few centimeters apart. The chessboard pattern is clearly visible on the IR camera, so we calibrate both cameras according to the method described previously. We refer to the RGB camera as camera 0 and the IR camera as camera 1. We take the chessboard pattern again and show it to both camera 0 and camera 1 at the same time. We detect the chessboard pattern in both cameras and recover the pose using

the previously determined camera matrices. Inverting the pose of the object gives us the pose of the camera relative to the object. The formula is as follows:

$$R_{camera} = R_{chess}^{T}$$

$$t_{camera} = -R_{camera} * t_{chess}$$

R_{chess} is the rotation matrix that gives the orientation of the chessboard relative to the camera and t_{chess} is the translation from the origin of the camera. Inverting this gives us the position and rotation of the camera relative to the chessboard.

Now that we have the poses of both cameras relative to the same object we can transform points between them. To go from an image coordinate in camera 0 to an image coordinate in camera 1 we take these steps:

- First we take a 2d image point in camera 0 and reproject it to a 3d point using the previously obtained camera matrix.
- Then we convert the 3d point in camera space 0 to world coordinates using the pose of the camera we found earlier.
- The world coordinate is then transformed back into camera space coordinates of camera 1 using its respective pose.
- We then project the 3d object point back to a 2d image point using the camera matrix of camera 1.

The conversion of a point v_0 in camera space 0 to a point v_1 in camera space 1 has the following formula:

$$v_{world} = R_0 * v_0 + t_0$$

$$v_1 = R_1^{T} * (v_{world} - t_1)$$

To visualize this process we first find both poses and then attempt to outline the chessboard by detecting it in one camera view and drawing it in both views using the transformation. An example of this can be seen in Figure 5.1. From this figure we can also see a small error, the chessboard is not aligned as well as in the color view. This misalignment is more noticeable under certain angles.

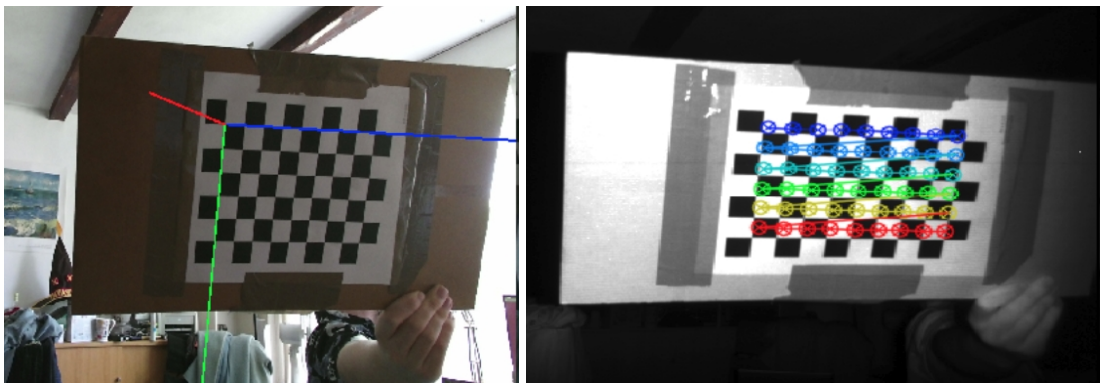


Figure 5.1: Chessboard detected in RGB camera and transformed to IR camera, Kinect 0

In Figure 5.2 we can see the same process repeated for the second Kinect. Which is also a transformation from the RGB camera to the IR camera. The error is much larger here. The process of calibrating the cameras was the same.

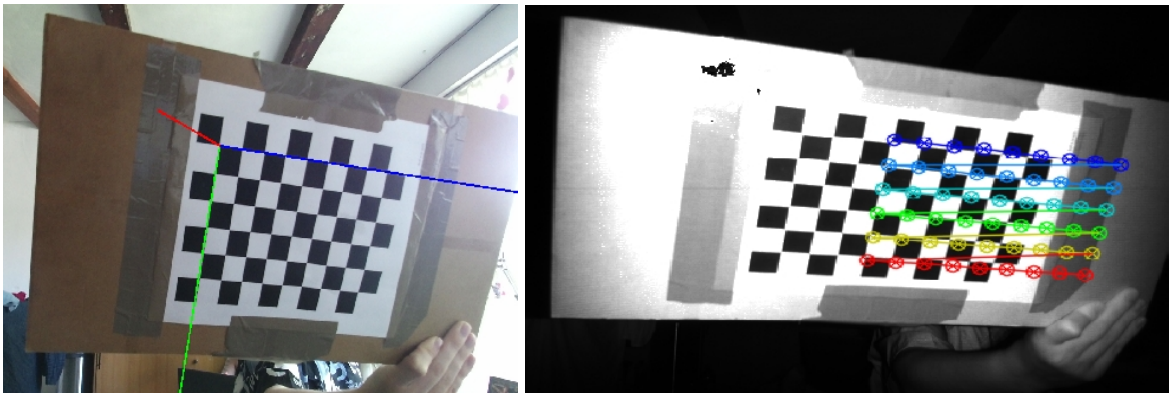


Figure 5.2: Chessboard detected in RGB camera and transformed to IR camera, Kinect 1

We also attempted to repeat this process for the RGB camera of Kinect 0 to Kinect 1 and similarly the IR cameras. Especially transforming between two cameras on different Kinects caused very large errors. With proper refinements to the method we can most likely minimize the error to a certain degree and make transforming between two Kinects more reliable. However the original aim of the dual Kinect setup was to improve body tracking accuracy, this method seems to only introduce more noise into the system. Therefore we decided not to pursue this method further.

5.2 Avatar-based Homography

The second approach we took was to base our transformation on a larger object. The chessboard is relatively small. A larger object will make the transformation more stable. The object we chose, the avatar, is already native to the Kinect. The Kinect extracts an avatar with 3d points for all joints in its camera space. We take corresponding avatars from both streams and save their joint positions. We also identify the stable joints, which are joints that are easy to detect by the Kinect. This should reduce the error caused by misdetection of the Kinect.

We then use the RANSAC algorithm introduced by Fischler and Bolles [FB81] to calculate an affine transformation matrix between the points. Since both coordinate systems have the same scale and are undistorted, affine transformation should have enough degrees of freedom to capture an accurate transformation.

When following these methods the transformation was still not accurate enough. We decided not to pursue this method further.

5.3 Avatar overlapping

Another problem with the before mentioned methods is that each time the Kinects are setup we need to estimate the homography again. Any movement of the Kinects will also introduce new errors. There was a need for a more practical solution. We decided to instead record both streams separately and not calculate an exact transformation between the two camera systems.

After both streams are recorded we form pairs of avatars at regular intervals from both streams. For each avatar we also calculate the relative position to its own root. In this case the torso joint as defined by the NiTE2 framework. For each pair we run a gradient descent algorithm that calculates a quaternion to rotate the avatar in stream 1 to overlap with the avatar in stream 0. Our loss function in this case is the distance between each joint in the avatars after rotation, which is then squared and summed. After we calculate a rotation for each pair we average all of them and determine an average rotation between the avatars. This way we can later rotate the paired avatar to match the other and take an average of the joint positions. This method will help reduce some large errors that either of the Kinects could make when detecting the avatars.

Chapter 6

Movements

6.1 Movement Selection

In order to run our experimentats we need a body of interesting motion data. We distinguish the following characteristics for this data:

- The movements should be cyclic so it can be looped indefinitely. One of our data features is the phase the animation is currently in. We are interested to see if this serves as a good heuristic for the network.
- The movements should cover a wide variety of motion in all of the joints in the body. This is to show that potentially any kind of movement could be learned.
- The movements should have different levels of complexity. The network should be able to handle movement throughout the body without being overwhelmed. For example hopping left and right while punching.
- The motions should have a variety in length. We want to explore the maximum length of movement that the network can learn. For example a dance of varying length. Or more simple but perhaps more confusing to the network a pattern of punches. Instead of just left-right alternating, left-left-right repeat.

There also exist some limitations when recording the motions. We cannot record motions that do not fit in the observed area of the double Kinect setup and they need to be recognizable by the NiTE2 driver. The Kinect is, for its price, a powerful tool to record motions. But it is certainly not without its limitations. For more details see Chapter 4.

Keeping the limitations in mind we have decided on the following movements:

- Left-Right Punching (LR PUNCH); A very basic upper body movement. The actor simple punches the air in front of them alternating between left and right. This animation has foot contact throughout the movement. There is a lot of rotational movement of the root. The shoulders and hips have to twist to follow the punch. There is almost no translational movement of the root.

- Left-Right Step (LR STEP); The actor steps left, pauses briefly, then steps right. The movement is exaggerated by lifting the feet and knees high when stepping. This should help to differentiate between the feet touching the ground or hovering above it. This is important to accurately label the data for floor contact.

- Left-Left-Right-Right Step (LLRR STEP); A movement comparable to to the Left-Right step. However an extra step in the middle is added for extra complexity and length of the animation. Since the observable area is limited the steps need to be small.

- Jab-Jab-Hook version 1 (LLR PUNCH 1); Similar to the Left-Right Punch we have the Jab-Jab-Hook. The actor starts by throwing a left hand jab twice and then a more pronounced right hook. The goal of this movement is to learn more realistic movement and have a slightly longer cycle with a repeat movement. The actor is stationary in the lower body as much as possible.

- Jab-Jab-Hook version 2 (LLR PUNCH 2); The upper body does the same but the actor is now instructed to make realistic feeling leg movement. Hopping, like boxers do, is realistic but it becomes very hard in that case to define foot contact. The feet are naturally sliding. We compromise by stepping back and forth in between punches.

For each recording we try to have the actor move as natural as possible. Due to the limitations of the Kinect and NiTE2's skeleton tracking the movement still needs to be exaggerated. We also need to take care to not position the arms too much in front of the body and make sure to not turn away from the cameras too much.

6.2 Combining the two views

We record from two Kinects at once. First we individually smooth the data with a three wide sliding window centered around the target value. This helps in removing some of the jittering that seems to naturally occur in the recordings.

We then need to combine the Kinect data into a single stream. We do this by first creating a list of pairs. We define a target frame rate for the final data stream. For each point in time we pick the data points that are closest from both streams. We take care not to select the same datapoint twice. We want as regular intervals as possible. In our testing an frame rate of 5 gave consistent unique datapoints from both streams. Each stream has its own coordinate system. The Kinects coordinate system has its origin in the depth camera and its Z axis extends outwards from the lens. We need to have corresponding coordinates from each system to arrive at a single avatar. After attempting multiple approaches we landed on a practical system of combining the streams. This process and how we arrived at it is explained in Chapter 5.

6.3 Data labeling

Each recording is reviewed manually. Often the software makes large errors when tracking the body. These errors are not recoverable and would greatly impact the quality of the input data and the result. These

recordings are rejected. An exhaustive list can be found in Chapter 4. For foot contact labeling the recordings are inspected frame by frame. The user can zoom in and rotate around the avatars for better inspection. We place labels for when the left or right foot are in contact with the ground.
We also assign labels for when the animation reaches a certain phase. For example in the left right punch animation the left arm fully extended marks the start of the animation and the right arm fully extended marks the half way point of the animation. The phase labels are then interpolated to fill the entire animation. Here follows a detailed description of what is determined as halfway point, or more precise, of the movement:

- Left-Right Punching; We label the first frame the left arm is fully extended as the start of the animation with phase 0. We label the frame the right arm is fully extended as π.
- Left-Right Step; We label the first frame of the animation as 0. The actor is still in the rightmost position with both feet on the ground. The first time both feet touch the ground and the body is straight in the leftmost position we label as π. When the actor is in the rightmost position again with both feet touching the ground we have completed one cycle.
- Left-Left-Right-Right Step; Similar to Left-Right step. We still label the rightmost position as 0 and leftmost as π. Just in this case an in-between step is inserted. We do not label this intermediate state.
- Jab-Jab-Hook 1; This motion is a little more complex. We would prefer to label the left punches individually. As a result of the Kinect not always accurately capturing the actor's movements the individual left jabs are not always easy to tell apart. We label the completion of the second left jab as the start of the animation with phase 0. We label the completion of the right hook as the animation half point with phase π. When labeling in one pass these are the points in time we deemed as most consistent and recognizable.
- Jab-Jab-Hook 2; We label this motion the same as Jab-Jab-Hook 1. The added steps are in line with the punching movements and do not change the way we want to interpret the data.

6.4 Floor plane estimation

The NiTE2 driver has built-in functionality to detect the floor. It returns a point in the camera space and a normal vector of the plane. However in our testing the floor plane was incorrect and feet were detected as not in contact with the ground when they were. We employ two strategies to calculate the floor plane ourselves. One is designed for motions that are in place and one is designed for motions that carry movement along the floor plane.

The static version is fairly simple since the normal vector does not need to be as accurate and the point on the plane is always close to the feet. We construct the floor plane via the point-normal form. We take the average of the foot positions as the point on the plane. For the plane normal we use the normal vector supplied by NiTE2.

The dynamic version is more complex. The plane normal needs to be more accurate since when the avatar moves away from the point on the plane any errors in the normal vector will become more apparent. For this

we use another way to calculate a plane. We find a set of points that exist on the plane and then fit a plane to these points. The points we use in this case are the foot contact points we defined earlier. For some recordings the foot contact points exist on a line more so than a plane. The resulting plane will have an incorrect normal vector perpendicular to this line. We inspect the result manually and in this case we use the static version as a fallback method.

6.5 Avatar features

Before we explain each feature and how it is calculated we need to set some definitions.

The avatar consists of 15 joints with connections to each other. The avatar's root position is the root joint position in the world coordinates. The root joint here is the Torso joint as defined by NiTE2. The root orientation is defined as follows:

$$v_{right} = \frac{(v_{left\ shoulder} - v_{right\ shoulder}) + (v_{left\ hip} - v_{right\ hip})}{2}$$

$$\hat{v}_{right} = \frac{v_{right}}{||v_{right}||}$$

$$\hat{v}_{up} = n_{floor}$$

$$\hat{v}_{forward} = \hat{v}_{right} \times \hat{v}_{up}$$

We obtain the three normalized directional vectors, $\hat{v}_{right}$, $\hat{v}_{up}$, $\hat{v}_{forward}$ as described above. Here n_{floor} is the plane normal of the floor. $_{left\ shoulder}$, $v_{right_shoulder}$ and so on are joint positions in the avatar.

Each avatar A has a root orientation and position as described above. It also has an index in the data stream A_i. The initial avatar is A_0. Every avatar is a collection of joint positions. $A = \{v_0, v_1, \ldots, v_{14}\}$. All indices correspond to a specific joint.

0	Head	8	**Torso**
1	Neck	9	Left Hip
2	Left Shoulder	10	Right Hip
3	Right Shoulder	11	Left Knee
4	Left Elbow	12	Right Knee
5	Right Elbow	13	Left Foot
6	Left Hand	14	Right Foot
7	Right Hand		

Table 6.1: Joint indices corresponding to joint names

We choose the torso as the root joint and it is therefore also the center of any rotation around the root.

For each combined avatar A_i in the stream we determine a previous combined avatar A_{i-1} to compare against. Since we have a frame rate of five this avatar is 200ms behind.
For each combined avatar we also rotate the up vector v_{up} to align with the world up vector. We calculate this rotation q as follows:

$$v_{axis} = \hat{v}_{up} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad \theta = \hat{v}_{up} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad s = 1/\sqrt{(1+\theta)*2} \qquad q = \begin{bmatrix} s/2 \\ v_{axis}.x * s \\ v_{axis}.y * s \\ v_{axis}.z * s \end{bmatrix}$$

The first feature we extract from the avatars is the relative positions of the joints. This is calculated by centering the root of the avatar on the world origin and rotating the avatar's forward to align with the positive Z axis. We use the same method as described for aligning the up vector.

We also calculate the avatar's relative joint velocities. We take the relative positions of the previous avatar A_{i-1} and the current avatar A_i and calculate the difference. We then scale this vector's length by the time delta between the two avatars. This gives us the joint speed per millisecond relative to the root.

The next few features are related to the avatar root movement. We discern the relative y-rotation, the absolute y-rotation, the relative translation and the absolute translation.
The relative rotation is the rotation around the Y axis in the clockwise direction from the previous avatar's forward to the current avatar's forward. We divide this again by the time delta to obtain the y rotational velocity per millisecond.
The absolute rotation is the rotation around the Y axis in the clockwise direction from the first avatar A_0 its forward to the current avatar A_i its forward.
The relative translation is the translation from the previous avatar's root to the current avatar's root. We then rotate this by the inverse of the forward direction. This makes the translation relative to the current root. We scale it by the time delta to obtain the relative translational velocity per millisecond.
The absolute translation is the translation from the initial avatar A_0 its root to the current avatar A_i its root. We then rotate this by the inverse of the forward direction of the initial avatar.

We include the foot contact labels for both feet. The final feature is the phase the animation is currently in. The total size of the feature vector is 100.

6.6 Data augmentation

To augment the dataset we apply a scaling factor to the bodies. The scaling factor goes from 0.9 to 1.1. We scale the bodies equally in the x, y and z dimensions. We regularly space our scaling factors to obtain 10 variants. We also apply noise to the dataset. We add a random value between -50 to +50 to the data. We do this twice for each recording to get pool of 20 variants total. Note that we never add any noise to the output. These augmentations should help stabilize the net when it encounters values slightly outside the expected range.

Chapter 7

Performance Metrics

In our experiments we focus on objective evaluation of the results. We differentiate between two types of evaluation. One type is a mix of custom and standard metrics that compare a prediction with its expected result. The other type we study is the long term performance of the network. We feed the output of the network back as the input.

7.1 Direct performance metrics

By default we track the performance of the network via Mean Squared Error (MSE) and Mean Average Error (MAE). However, by the nature of the data, not all outputs of the network are equal in their effect on the resulting animation. Therefore we also define a weighted MSE function. This function values the rotations and translations of the root as well as the foot joint positions higher than the rest of the features. A small error in the root orientation or position can cause major problems with the animation playback. Even if the rest of the network is performing well.
We also track the accuracy of the foot contacts directly. A foot contact means the bottom of the foot of the avatar is in contact with the ground. The foot contact data is hand labelled. Foot contact is an important feature in this setting because it enables us to calculate foot sliding. Knowing the accuracy of the foot contacts allows us to give context to the results of the foot slide error as well.

Our main metric in the experiments is foot slide error. We calculate foot slide error by comparing the last input of the network with the output. We construct the avatars from these feature vectors and check if there is a matching pair of positive foot contacts. For example when the left foot touches the floor in the last frame as well as in the current frame. We then calculate how much the foot moved despite being in contact with the ground. This error is then squared and averaged with the other foot in the case both feet were touching the ground.

Another metric we track is the limb length error. This metric is designed to check the avatars structural

soundness. Limb lengths should ideally be constant. We initially calculate the average of the limb lengths in the validation data. For each prediction in the validation data the limb lengths are then calculated as well and compared to the average, we take again a MSE.

We mentioned that we value some features higher than others since they impact the final result more. This is however an approximation of the true error that is made in the avatars. With the total body MSE we construct two states of the avatar again. One based on the prediction and one on ground truth. Their joint coordinates are directly compared and we can calculate the true error as a distance between the joint positions.

The last metric we define is total body energy. We calculate the amount of movement or energy that exists between the avatar based on the last input and the prediction. This is mainly an indicator that we use in the long term performance evaluation. We need a baseline to see how much energy is normal in the animation.

7.2 Long term performance metrics

The second type of performance metrics we specify evaluates long term performance. We give the network a start point and then we monitor the performance of the network when we feed the output of the network back into the input. So if we have n timesteps of our RNN the nth timestep shifts to the $n-1$ position, $n-1$ to $n-2$ and so on. We fill the previous prediction in as the new nth timestep. The goal of these metrics is to evaluate the long term stability of the animation, as well as the overall performance.

Similarly to the direct performance metrics we monitor the foot slide error (FSE). We construct the animation frame by frame and measure the error over the course of 60 seconds. We monitor the MSE over the first 5 seconds, 10 seconds, 20 seconds and 60 seconds. This is to track if the performance degrades over time.

The limb length error (LLE) is also tracked over time. In this case we calculate the limb length averages from the given start point. We run the animation for 5, 10, 20 and 60 seconds and track the MSE made on the limb lengths.

We monitor the total body energy (TBE) over the first 60 seconds as well. This gives us an indication of how the animation behaves over this amount of time. We calculate the average energy over the validation data first to set a baseline. Now we can detect some performance issues quickly. For example if the energy level bottoms out this indicates a deadlock. If it is very high the animation is most likely chaotic or distorted. A good quality cyclic animation will also show a cycle pattern in the energy.

The Body Matching Error (BME) is defined as the minimum MSE of the predicted avatar relative to the root compared to any of the relative avatars in the test data. When comparing the avatars the net produces to the avatars in the training set we first need to make them face forward again. This is because even though the net predicts relative joint positions they can still build up errors over time. This causes the produced relative avatar to not face exactly forward again and increases errors unjustly. This process does increase the dependency of this metric on the joints used to center. In our testing we did see that the BME dropped significantly when the

correction was applied. So we believe this is an improvement to the metric. The test data should contain all expected avatar poses the net can produce.

To validate the performance of the metrics themselves we also have the option for visual inspection. We can play a variable amount of animation frames and inspect the result.

Chapter 8

Network

With the data prepared and defined we need neural nets to learn it. We made the choice to configure and train a separate net for each movement. We do this by first defining the hyperparameters that we want to optimize and then perform a three step grid search hyperparameter optimization.

8.1 Grid search strategy

We define the following parameters and constraints for our network. Firstly any of the nets will have a set of at least one recurrent layer, followed by at least one dense layer.
The data is generated to have 5 time steps as input of the recurrent layer and is divided in batches of size 32. Our data is recorded at five frames per second, so the network gets one second of animation as input.
We use a learning rate schedule with an initial learning rate of 0.0001, a decay rate of 0.9 and 10,000 decay steps. The learning rate updates as follows then:

$$learning\_rate \; = \; intial\_learning\_rate \; * \; decay\_rate\hat{}(step \; / \; decay\_steps)$$

We employ an early stopping algorithm that will stop the training after not finding an improvement in the loss function for 10 epochs. We do not use an extra validation set here for the early stopping since we are more so interested in a fast assessment of the configuration and less so in finding the optimal fit at that specific point in time. We train for a maximum of 150 epochs. In our experiments we found that this is enough estimate the rate of convergence of the model.

We considered using the stateful option for the Recurrent layers. When training in stateful mode the hidden states of the recurrent layers are not reset between batches. We should take a batch size of 1 and disable shuffling of the data. We reset the states each epoch. This way the entire data set is recognized as one big sequence. However stateful Recurrent models only apply to datasets where there are strong long-term dependencies. These could very well exist in our movements but instead we opted to use the phase label as a

guide of position in the animation cycle.

We use the Relu activation function for the dense layers since it is an all-round good, fast converging and computationally cheap activation function [Aga18].

We search the following parameters for their optimum.

- Recurrent layer type. The options are default Recurrent layer, LSTM layer, GRU layer.
- Recurrent layer activation function. We propose tanh and relu
- Recurrent layer count. Our nets contain either one or two recurrent layers. These layers are of the same type and use the same activation functions.
- Recurrent layer node count. We try 1024 and 512. If we have multiple recurrent layers each consecutive layer will have the same amount of nodes.
- Dense layer count. Similarly to recurrent layer count we have either one or two dense, fully connected layers. These layers will always have the relu activation function.
- Dense layer node count. The first dense layer will have either 512 or 256 nodes. Any following layer will have the same amount of nodes.
- Dropout. Dropout is a method classically used to avoid overfitting. We randomly set an input to zero. We try probabilities 0.1, 0.2 and 0.3 for our training.
- Recurrent Dropout. Recurrent dropout is similar to Dropout but a method specifically used for recurrent layers. An input from a previous timestep is randomly set to zero. We use probabilities 0.1, 0.2 and 0.3.

These parameters combined create a sizable search space. All the parameters make for 432 combinations. For each combination we use 3-fold cross validation. With the implementation of grid search used we train three separate nets for each combination and average the validation scores. On top of that LSTM layers converge slower [HS97] on average so we want to minimize the search space.

We propose a strategy of dividing the search parameters into groups. We optimize the groups in order and use previously found optimums.

Stage 1
We decide on what type of recurrent layer and which activation function we use. We deem this the most fundamental parameters of the net. We have a total of 6 combinations here.

Stage 2
Using the optimal parameters of the last stage we now inspect the recurrent layer and node count. We also inspect the dense layer and node count. These parameters related to the width and depth of the net are a logical next step. These parameters give a total of 16 options.

Stage 3
In this stage we inspect the dropout parameters. Having found our optimal size and composition of the

network we look at the dropout settings. We have three options for both dropout variants which gives us a total of 9 combinations.
In our parameter search space we have no option for 0 dropout in either category. Networks without dropout converge much faster but have shown to be unstable in our testing. Mainly in the long term performance metrics we encounter the exploding gradients problem.

All stages combined give us a total of 31 combinations to explore.

8.2 Training results

When testing the different nets we found that some of our features were converging poorly and corrupted the animation. We include for each predicted frame of animation an absolute root rotation and a root rotation relative to the previous frame. Initially we averaged the new absolute root rotation and the relative rotation applied to the previous root. The relative rotation was on average too large and caused the avatar to rotate in an exaggerated manner. We observed that the animation quality improved drastically when the bias was put more towards the absolute rotation. Similarly for new joint positions we calculate by combining the next set of predicted joint positions with the previous joint positions and the velocity of each joint. We observed that the animation became more "smoothed out". When putting more of a bias to the joint positions the animation became sharper and more accurate to the original. We also looked into how the new root position is calculated. For the root position we predict an absolute position and a position relative to the previous position. We tried shifting the bias between these. There was no improvement so we opted to use the added robustness of two ways to predict the root position.

It must be noted that when playing back the training data it did not show these errors in the relative root rotation and joint velocities.

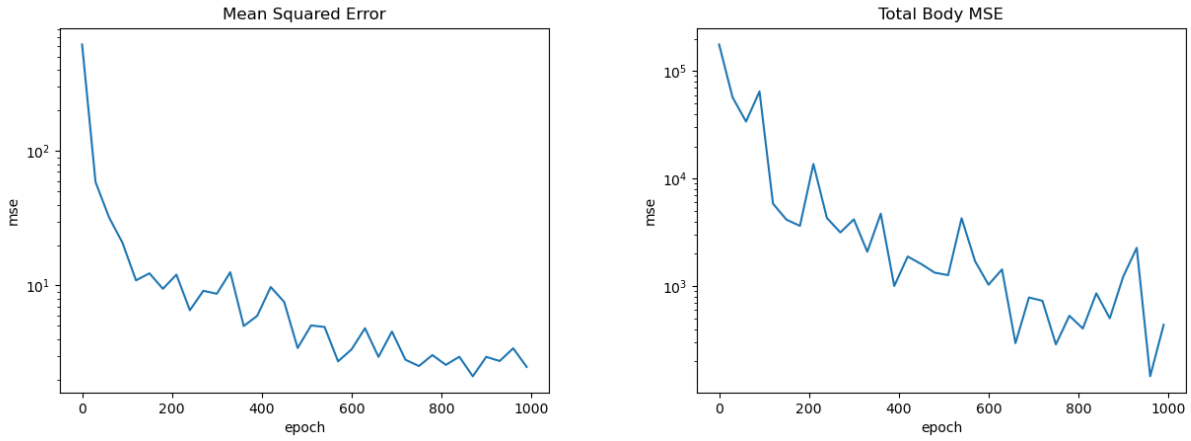
All hyperparameters are compared using a weighted Mean Squared Error loss function. The outputs related to the foot positions are weighted heavier as well as the root orientation parameters. We are focusing on foot sliding and want to express this in our loss function to optimize the network with an eye towards that. For all movements LSTM layers performed best on the trainings data. LSTM layers are commonly paired with either tanh or relu activation functions, in our testing the relu outperformed the tanh functions by a small margin. Intuitively for the layer counts and node counts of the recurrent and dense layers the highest amounts of nodes performed the best in our tests. To recall this is 2 layers of 1024 LSTM nodes and 2 layers of 512 dense nodes. From the gridsearch process we found that a lower dropout rate results in a better score. However we deemed dropout an essential part of network stability and decided to override the setting with 0.3 normal dropout and 0.2 recurrent dropout.

Chapter 9

Experiments

With our various metrics we want to establish an understanding of the naturalness of our animations. We run them for our various movements and compare some of the results.

9.1 Total Body Error



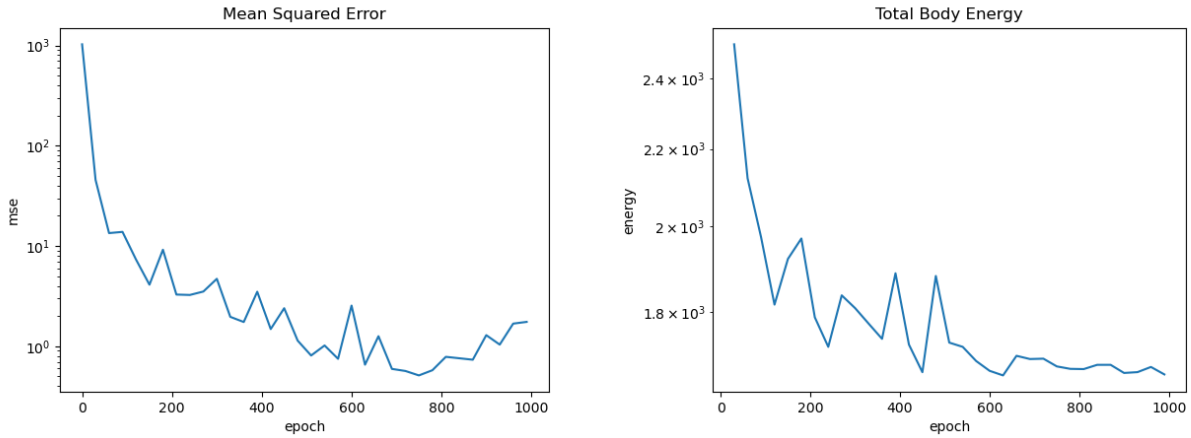
(a) Unweighted validation Loss

(b) Total Body Error

Figure 9.1: LR PUNCH, Total Body Error comparison

In Figure 9.1a we can see the MSE starts to bottom out at around epoch 500. When compared to Figure 9.1b we see that there are still improvements to be found after epoch 900. Also the graph of Total Body MSE fluctuates more strongly. This could be caused by the metric depending more on a select few features. When features related to root rotation/position contain errors they affect the entire body. While training we weighted these features stronger and this improved the stability of the animation a lot. The Total Body Error metric is a better approximation of the error compared to our weighted MSE and could potentially also be used as the loss function of the net. Using this function would be considerably slower however.

9.2 Total Body Energy



(a) Unweighted Validation Loss

(b) Total body Energy

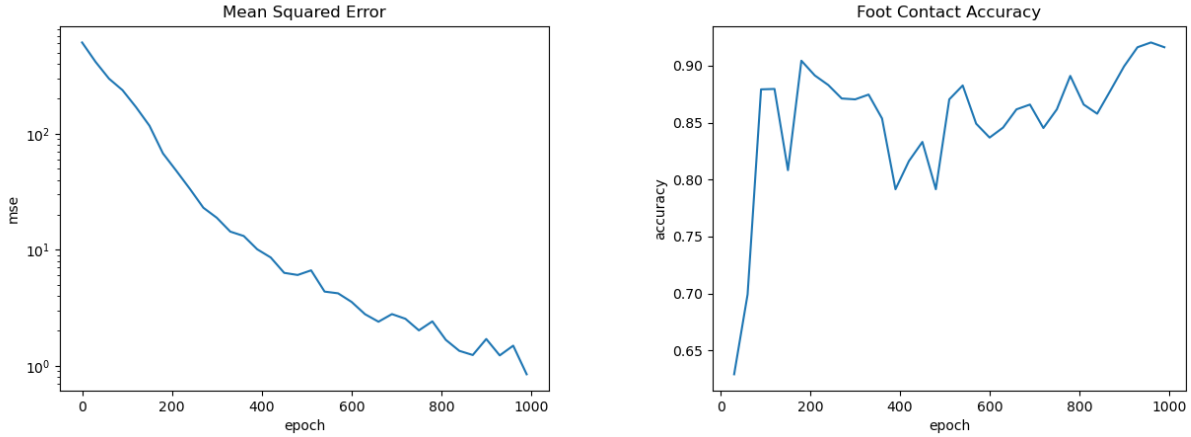
Figure 9.2: LLR PUNCH 2, Total Body Energy comparison

In Figure 9.2b we see the progression of the average energy. Through the epochs the error in the energy in the predictions goes down.
This indicates a combination of two things:

- Static joints in the animation move less.
- Non-static joints follow the intended movement better.

Furthermore some of the bigger peaks around epoch 200, 400 and 500 can be linked to the peaks in MSE as well, indicating some correlation.

9.3 Foot Contact Accuracy



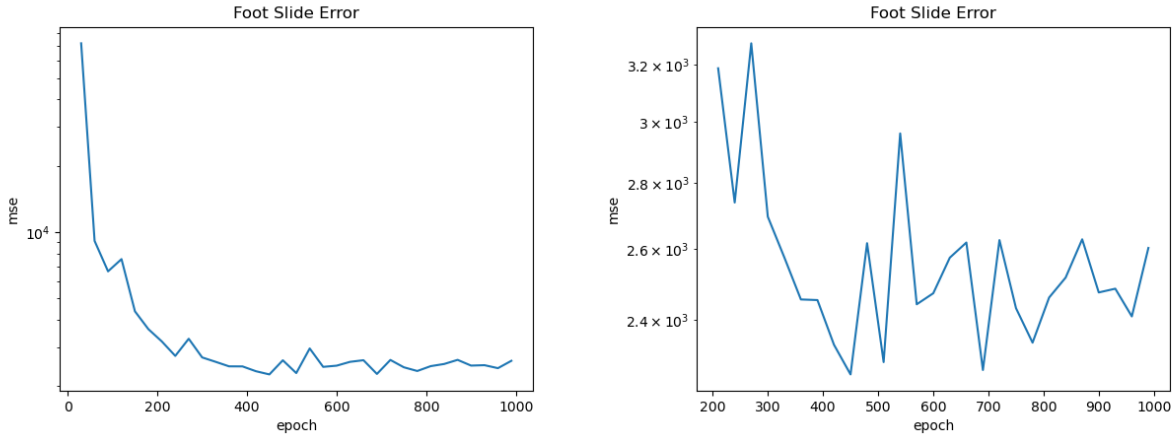
(a) Unweighted Validation Loss

(b) Foot Contact Accuracy

Figure 9.3: LLRR STEP, Foot Contact Accuracy comparison

The Left-Left-Right-Right Step movement was one of our best converging datasets, Figure 9.3a. The validation loss is a consistent drop during the entire training. The Foot Contact Accuracy does not converge as smoothly but reaches an acceptable 0.91 accuracy. We need a high foot contact accuracy for the long term Foot Sliding Error.

9.4 Foot Slide Error



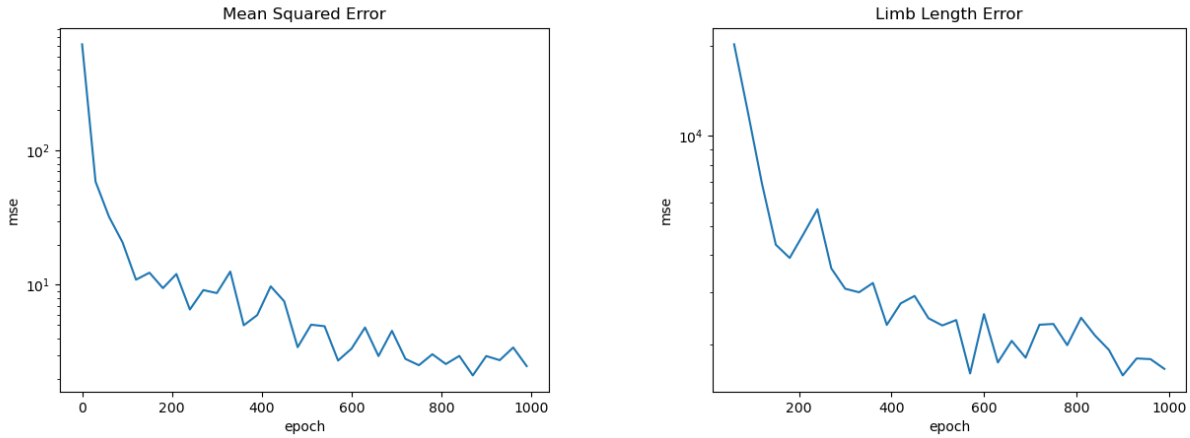
(a) Foot Slide Error

(b) Foot Slide Error

Figure 9.4: LLRR STEP, Foot Slide Error comparison

In Figure 9.4a we see the convergence of the Foot Slide Error. In the final epoch we have a foot slide of roughly 2600. This initially seems high until we realize that the ground truth data contains an error of roughly 2500. Also we can see some early optimums around epoch 450 and 700. Although these could be better weights we also have to take into account other factors such as the stability of the net when generating longer sequences of animation. Furthermore it is possible the feet are simply moving less and therefore creating less slide.

9.5 Limb Length Error



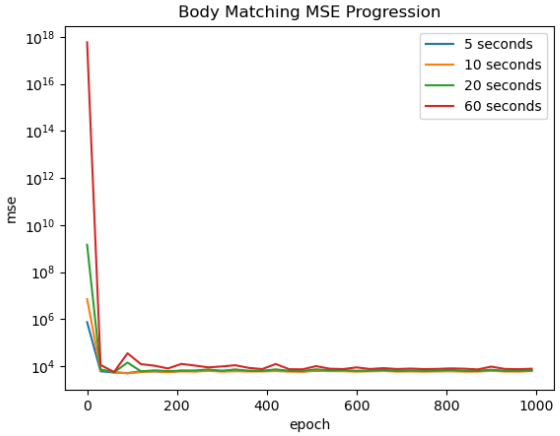
(a) Unweighted validation Loss

(b) Limb Length Error

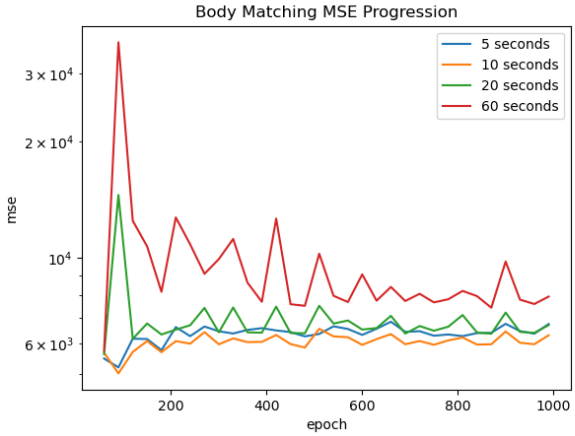
Figure 9.5: LR PUNCH, Limb Length Error comparison

Limb Length Error is a good metric to test on the Punching movements since the arms tend to over extend or not extend enough when punching, stretching the limbs in the process. The Limb Length Error decreases comparatively to the validation loss up till epoch 600 roughly. Afterwards both metric's improvements flatten out. The Limb length Error and MSE when compared in these graphs.

9.6 Body Matching Error, Long term

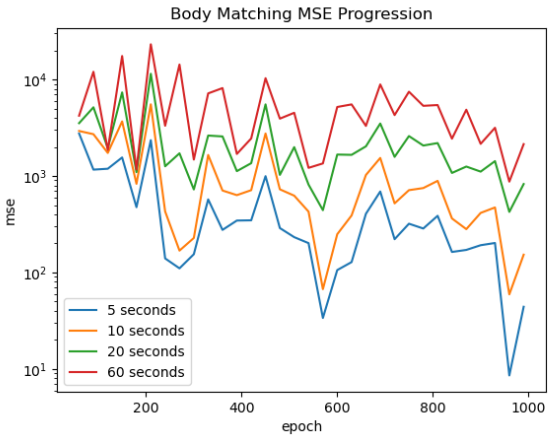


(a) Body Matching Error, LLR PUNCH 1



(b) Body Matching Error, LLR PUNCH 1, zoom

(c) Body Matching Error, LR STEP

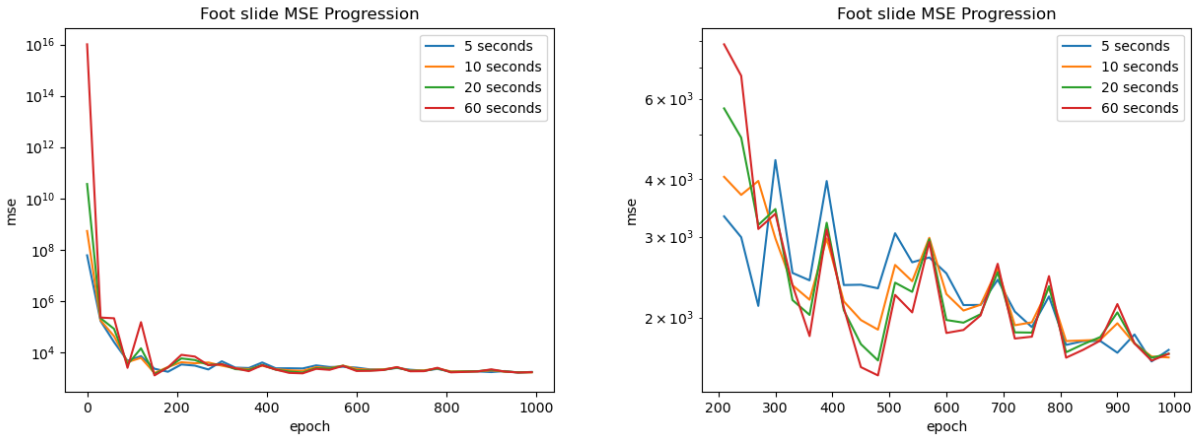


(d) Body Matching Error, LR STEP, zoom

Figure 9.6: LR STEP & LLR PUNCH 1, Body Matching Long term

We chose the Left Left Right Punching movement initially since it has relatively more poses compared to the other movements. In Figure 9.6ab we can see the progression of the error throughout the animation. An unexpected result is that in the first 5 seconds the error is higher on average than in the first 10 seconds. This could indicate that the first cycle of the animation is unstable and the animation stabilizes in later cycles. In comparison we have the same metric for the Left Right Step movement in Figure 9.6cd. For this movement the error builds proportionally in relation to the time it is running.

9.7 Foot Slide Error, Long term



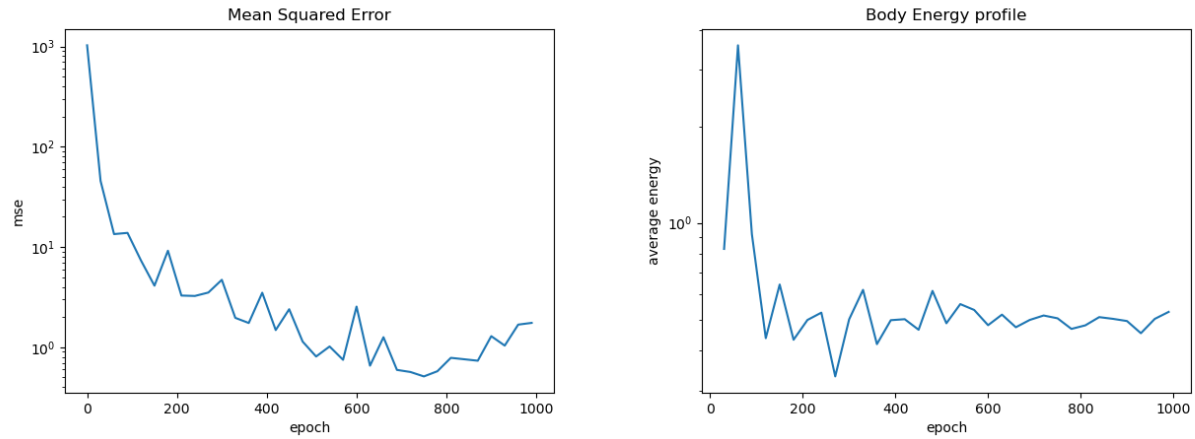
(a) Foot Slide Error, LLRR STEP

(b) Foot Slide Error, LLRR STEP, zoom

Figure 9.7: LLRR STEP, Foot Slide comparison

The Foot Slide Error shown in Figure 9.7ab has a big improvement over the epochs. It is remarkable that around epoch 400, 580, 700 790 the foot sliding error is almost the same for each duration. Also for many of the durations the foot sliding error decreases when the animation is played longer. In the worst case this could indicate the animation reaches a deadlock after a certain amount of time. At the final epoch the foot sliding error seems to be stable since the error does not increase or decrease significantly for longer durations.

9.8 Total Body Energy, Long term



(a) Unweighted validation Loss

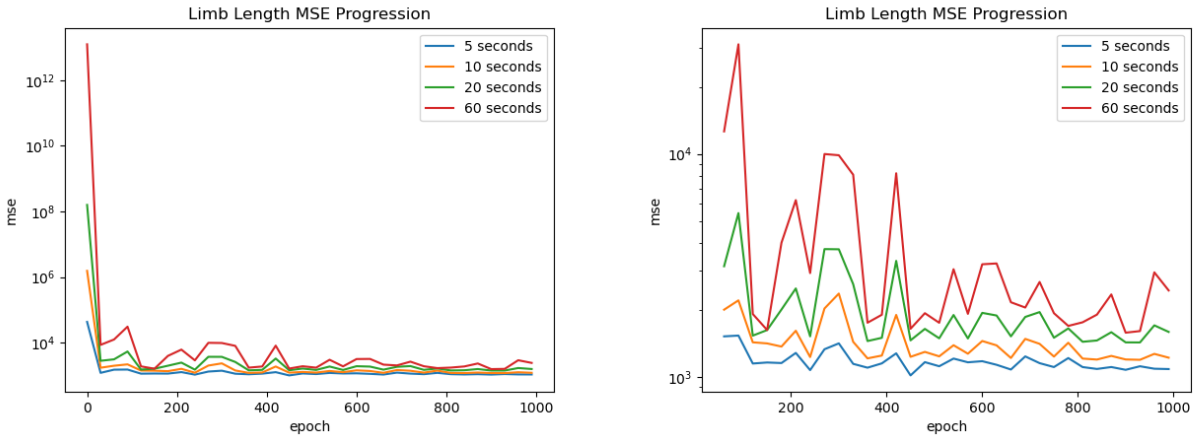
(b) Body Energy normalized

Figure 9.8: LLR PUNCH 2, Body Energy comparison

The Total Body Energy in Figure 9.8b is normalized against the average body energy of the datasets. Notable is that the energy is on average lower than in the training data. This could mean the avatar is not following

this motions or the animation is played more smoothed out. For example a punch not fully extending or being as snappy as in the training data. If the energy drops lower it could indicate a deadlock in the animation.

9.9 Limb Length Error, Long term



(a) (b)

Figure 9.9: LR PUNCH, Limb Length Error

In Figure 9.9ab we have the progressions of the Limb Length Error. We show the MSE over the first 5, 10 20 and 60 seconds of animation. This error increases proportionally with the animation time. Around epoch 800, 900 and 950 we can see that the error did not increase as much for longer animation duration. This could indicate a stable animation that does not degrade over time.

	Short Term						Long Term		
	val MSE	val MAE	CA	FSE	LLE	TBE	FSE	LLE	BME
LR PUNCH	2.48	0.58	1	2700	1700	430	2100	1600	670
LR STEP	0.46	0.33	0.99	5900	1400	70	5600	870	830
LLRR STEP	0.84	0.47	0.92	2600	470	130	1700	440	89
LLR PUNCH 1	1.48	0.64	1	5700	2500	131	2500	1600	6700
LLR PUNCH 2	1.75	0.62	1	7000	1900	77	5500	2700	4900

Table 9.1: Overview of all metrics

We summarize the results of all metrics in Table 9.1. We show Mean Squared Error on the validation set (val MSE), Mean Averaged Error on the validation set (val MAE), Contact Accuracy (CA), Foot Sliding Error (FSE), Limb Length Error (LLE), Total Body Energy (TBE). The long term performance metrics are evaluated on 20 seconds of animation.

Chapter 10

Evaluation

We constructed an optimal net for each movement and want to evaluate the performance of our metrics. To do this we need to compair against a ground truth. We gave out a questionnaire to a test group in which they score the animation quality and naturalness.

The questionnaire is divided into 5 sections, one for each movement we study. Each section starts with a video. The video starts with reference footage of the movement in real life. Then we show three angles of the animation as constructed by the network. We display the animation on a simple stick figure avatar. We add some markers to give better context to the orientation of the avatar. We ask the same questions for each animation to establish a constant evaluation of the different movements.

How similar is the animation to the reference video? Rather than generating new locomotion we are interested in mimicking existing footage. Therefore this question gives us insight in to what degree we succeeded at that.

Does the animation degrade over time? Another important aspect of the animation is the stability. We show 20 seconds of animation in the video. This will contain at least two full cycles of animation. In the case of an unstable net we highlighted some of the following defects:

- Animation slows down or comes to a standstill. Sometimes the animation can reach a type of deadlock in the animation cycle. This generally happens if there is a point in the training data where the avatar is reversing the movement and stands still for a moment.
- Animation speeds up. We did not encounter this in our initial observations but for completeness it was also included.
- Avatar grows or shrinks in size. When the animation plays the avatar will in some cases grow or shrink in size. This was an problem when the augmented training data was skewed to have more enlarged avatars opposed to shrunken avatars. The net generalized this to always shrink the size of the avatars.
- Animation becomes chaotic. In our opinion this indicates failure to learn the animation. The animation

takes a shape that is outside the learned poses and grows more and more chaotic. It becomes unrecognizable as a human skeleton. We minimized occurrences of this problem by adding noise to the data so that poses that are slightly outside the poses in the training data can be denoized and recovered to stay within the known poses.

The participants could pick any amount of the defects if they deemed it existed in the animation.

How much foot sliding is observed from the front view? A main focus of this work is to minimize the foot sliding in the animations. Foot sliding is a common issue with animation that will instantly make the animation look less natural. We want to see how much the perceived foot sliding by the participants corresponds to the foot sliding we detect using our algorithm.

Does the animation "feel" natural? Most certainly the most subjective question in the questionnaire. We are interested in what our participants intuition says about the animation.

Additional comments In case we missed any defects or if the participants have anything else they notice about the avatar or the animation they can leave their remarks here.

10.1 Evaluation results

Our evaluation had eight participants of a wide age range (20 - 65). The participants were approached individually and had unlimited time to fill out the questionnaire. The participants reported to take on average 10 minutes.

How similar is the animation to the reference video?

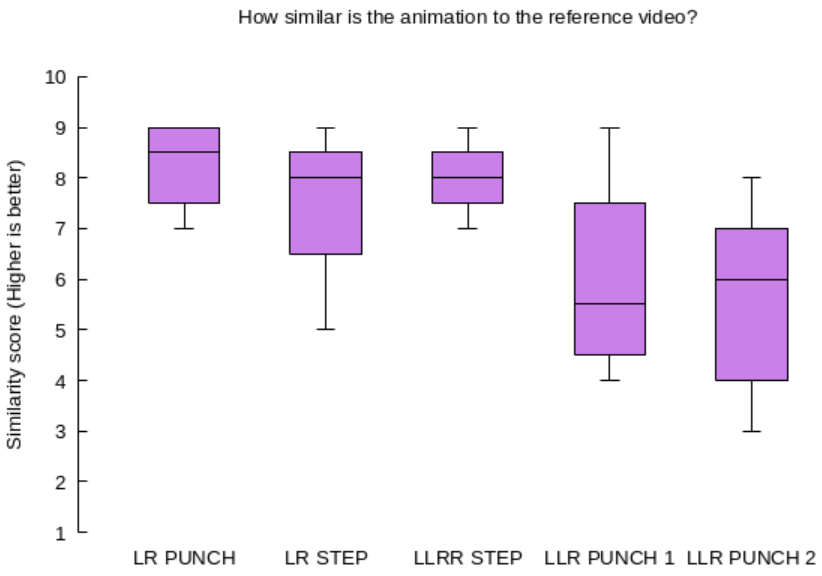


Figure 10.1: Question 1 Summary

The movements were speculatively sorted in the questionnaire by their complexity. In Figure 10.1, we see that the similarity score as rated by our participants goes down for each movement. The LR PUNCH and LLRR

STEP score well on this test. The scores of LLR STEP 1 and 2 have significantly more variance in their scores and score on average lower.

Does the animation degrade over time? The majority of participants, 87.5 to 100 per cent, agreed the animation quality stayed the same throughout the recording and did not degrade. For this questionnaire we showed the first 20 seconds of the animation, it would be interesting to also review the first minute or more and see if the animation degrades then. However since the participants had to review five different movements we concluded 20 seconds per animation from three perspectives would be the limit.

How much foot sliding is observed from the front view?

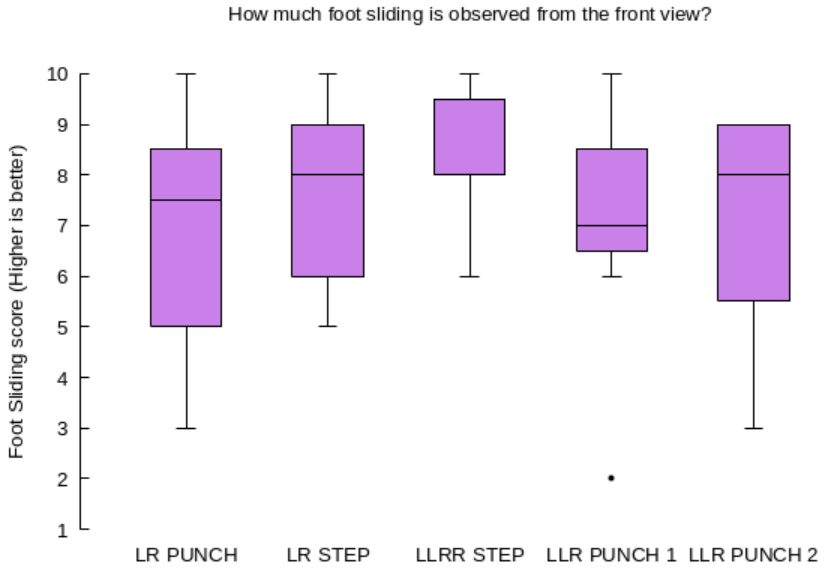


Figure 10.2: Question 3 Summary

The scores give for foot sliding vary much more for each movement compared to question 1, Figure 10.2. The description given for a score of 1 was: "No sense of contact". So it feels like the avatar is not grounded at all. The description for a score of 10 was:"No sliding". The high variance would indicate that it is hard for people to agree on the amount of foot sliding.

We specified that the foot sliding should be judged from the front view only. This is for the reason that the raw Kinect data is also most accurate when viewed from the front. Judging the distance of a joint to the camera is remains a problem of the Kinect. When viewing the Kinect data from above similar errors can be seen. When viewed from the front the data looks quite accurate. We want to keep the comparison equal, therefore we decided to only include the front view for this question.

Our best performing movement in this metric is LLRR STEP. The LR PUNCH does not score as well. Perhaps since the feet are known to be in contact with the floor for the entire duration of the animation it is more clear to the participants when the feet are sliding.

Does the animation ”feel” natural?

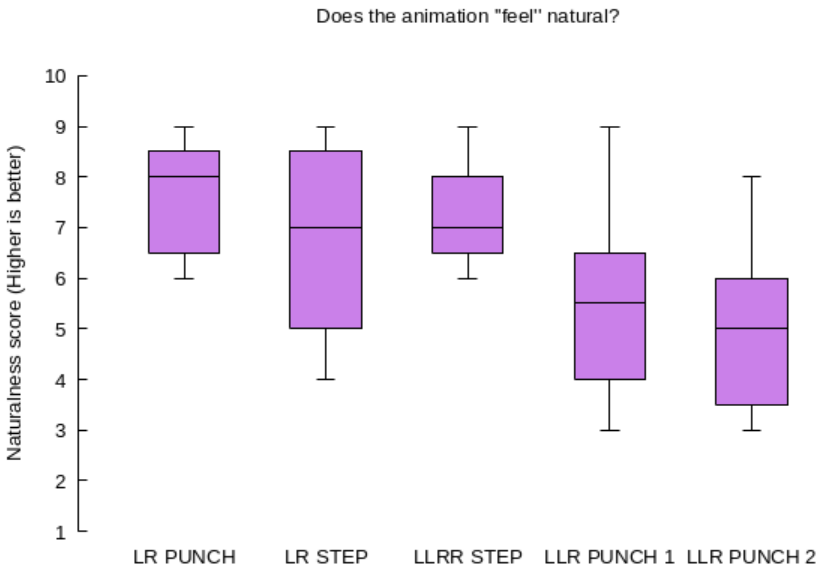


Figure 10.3: Question 5 Summary

Arguably the most subjective out of all the questions is the naturalness score, Figure 10.3. When we compare the naturalness score and the similarity score we can see they are comparable but slightly lower on average. Understandably the animation being more similar to the reference video would also make it more natural. The participants had perhaps higher standards for if an animation feels natural.

Additional comments Many of our participants noticed that the top view was significantly worse than the other views.

Some of the other comments we received were that the frame rate could be higher for improved clarity. The frame rate for this project is directly tied to the frame rate at which the recording was taken. However using linear interpolation of the joint positions we could make the frame rate arbitrarily higher. This naive approach would cause one issue. We can not interpolate joint positions that lay outside the predicted joint positions. For example if we do not predict the most extended point of the arm in the punching animation we could not fill this in as easily. Perhaps we could carry a momentum. Nonetheless it could have been an improvement to the project.

Chapter 11

Discussion

To validate our performance metrics we want to link them back to the different scores on the questionnaire and see if they correlate. This way we can also gauge what level of error corresponds to what level of perceived naturalness. For each metric we inspect the same duration of 20 seconds as the participants reviewed in the evaluation. As an objective measure of correlation we use the Pearson's correlation [Kiro8] and the Spearman's correlation [speo8].
The Pearson's correlation coefficient for variables X and Y is defined as follows:

$$pearson(X,Y) = covariance(X,Y)/(stdv(X) * stdv(Y))$$
$$covariance(X,Y) = \sum_{i=1}^{n}(x - mean(X)) * (y - mean(Y)) * \frac{1}{n}$$

The Pearson's correlation coefficient is an indication of the strength of the linear relation between two variables. It ranges from -1 to $+1$. A negative value corresponds to an inverse relation. The closer the value is to 0 the lower the correlation between the variables. Anything below -0.5 or above 0.5 is regarded as a notable correlation. The Pearson's correlation is easy to interpret and therefore a good choice.

The Spearman's correlation coefficient is another indicator for the relation between variables. However the relation does not have to be linear in this case. The Spearman's correlation coefficient is a good choice when the distribution of the variables and possible relation between two variables is unknown. The Spearman's correlation coefficient is defined as follows:

$$spearman(X,Y) = covariance(rank(X), rank(Y))/(stdv(rank(X)) * stdv(rank(Y)))$$

11.1 Similarity score

We are interested to link our Body Matching Error to the similarity score. The BME compares the predictions to the avatar shapes in the input data. This is closest to how the participants rated the similarity score in the evaluation.

Movement	Similarity score (mean)	Body Matching Error (20s mean)
LR PUNCH	8.25	668
LR STEP	7.50	825
LLRR STEP	8.0	89
LLR PUNCH 1	6.0	6708
LLR PUNCH 2	5.65	4858

Table 11.1: Similarity Score datapoints

In Table 11.1 we can see the different datapoints for the similarity score. We have a range of scores between 5.7 and 8.3. This is out of the total range of 1 to 10, this is an adequate fraction of the total range.

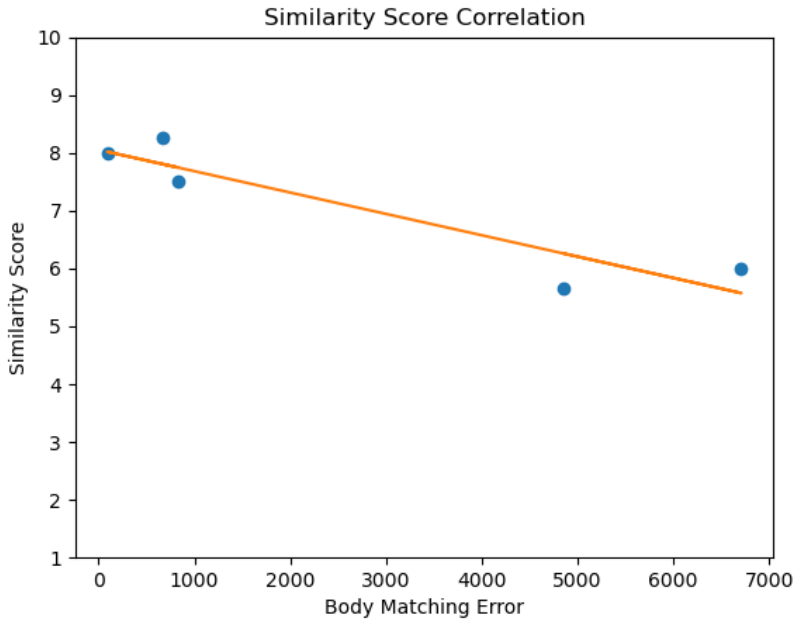


Figure 11.1: Similarity Score Correlation

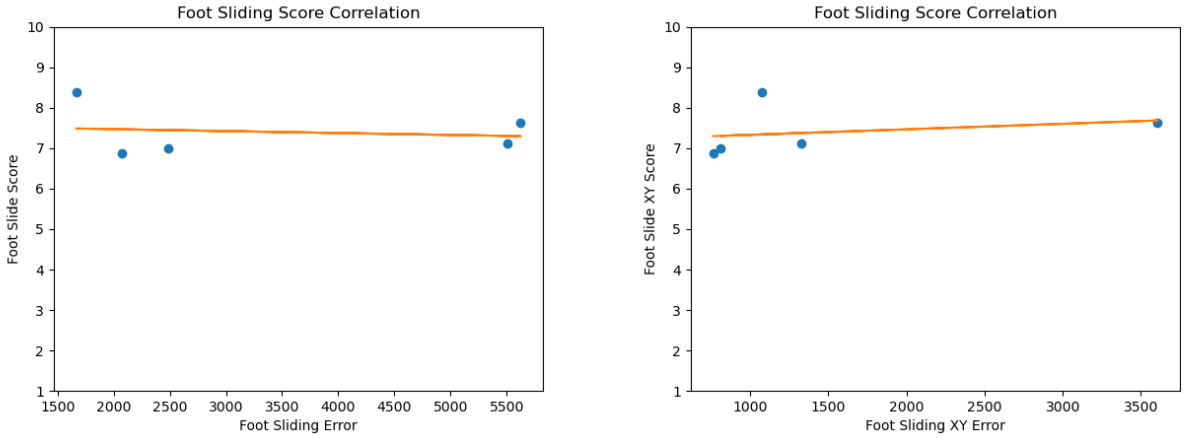
We visualize the datapoints by plotting them as seen in Figure 11.1. We can see the points fit quite well on the line. We also apply a quantification of the correlations. We found a Pearson's correlation of -0.92, which indicates strong linear correlation. For the Spearman's correlation we found a value of -0.8, which also indicates a strong correlation. From our tests it seems that the Body Matching Error is a good predictor for the similarity score.

11.2 Foot Slide score

Movement	Foot Sliding score (mean)	Foot sliding Error (20s mean)	Foot sliding Error XY (20s mean)
LR PUNCH	6.88	2075	763
LR STEP	7.63	5625	3607
LLRR STEP	8.38	1665	1076
LLR PUNCH 1	7.00	2488	812
LLR PUNCH 2	7.13	5504	1325

Table 11.2: Foot Sliding Score datapoints

Table 11.2 contains the datapoints for the Foot Sliding score. In this case our range is smaller compared to the Similarity score, the range is 6.9 to 8.4.



(a) Total Foot Slide Correlation

(b) XY Foot Slide Correlation

Figure 11.2: Foot Sliding Score Correlation

When inspecting the distribution of our points and the regression line in Figure 11.2a, we see that the variables are most likely weakly correlated. The regression line has a low slope. The corresponding Pearson's correlation coefficient is -0.15. The relation is inverse as expected. However the absolute coefficient is below 0.5 indicating a weak correlation. The Spearman's correlation coefficient is 0.0 meaning no correlation. We asked the participants to only consider the front view for the evaluation of the data. This means we only observe the foot slide in the XY plane. Our metric of Foot Sliding Error takes foot slide in all dimensions into account.

We run our foot slide metric again, this time we only calculate the Foot Slide Error in the XY plane. The results are shown in the rightmost column of Table 11.2. We plot our findings in Figure 11.2b, the result is similar. The datapoints are even closer distributed and our Pearson's score is now 0.26. The absolute value increased by 0.11 indicating a stronger correlation. The Pearson's correlation has become positive however. The relation between the error and foot sliding score should not be positive, a lower FSE should give a higher Foot Sliding Score. The Spearman's correlation coefficient is 0.7. This is a big increase of 0.25. Again the value is positive

indicating a positive relation between the variables. These results indicate that adjusting for the viewer's perspective did not improve the performance of the metric.
When reviewing the animation it is a challenging task to accurately assess the level of foot slide. Especially in the given review format. For this task displaying the animation on a more complex avatar with a mesh would be more intuitive for the participants. Another issue is that although our Foot Contact Accuracy metric is high (0.91 to 1.0), we have no guarantee as to how it degrades when the animation is played. For this we would have to train a separate net that evaluates for a given avatar whether its feet are in contact with the floor. Such work has been done by Ma et al [MYYL19]. We theorize that these adjustments would improve our metrics performance significantly.

11.3 Naturalness score

Movement	Naturalness score (mean)	Limb Length Error (20s mean)
LR PUNCH	7.625	1592
LR STEP	6.75	867
LLRR STEP	7.25	435
LLR PUNCH 1	5.5	1600
LLR PUNCH 2	5.0	2732

Table 11.3: Naturalness Score datapoints

The datapoints for the naturalness score as seen in Table 11.3 however cover a good range of 5 to 7.63. The Limb Length Error's range is similarly diverse. These are promising qualities.

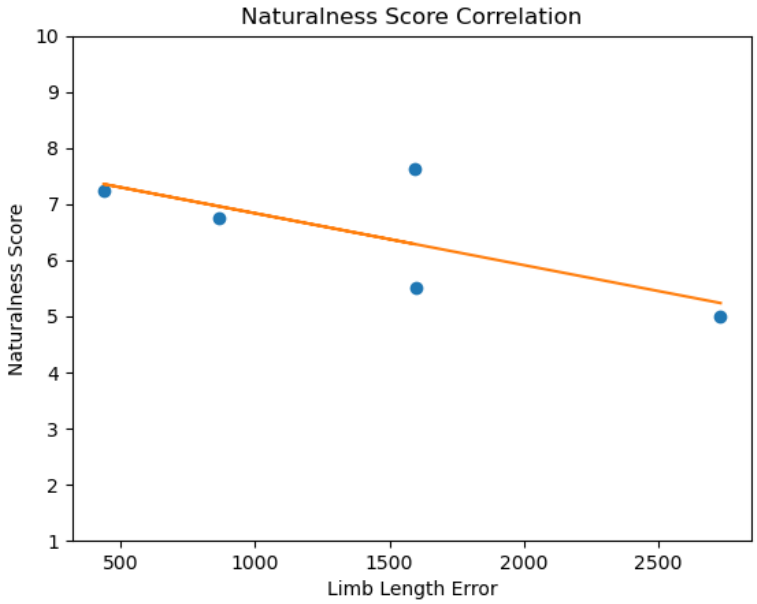


Figure 11.3: Naturalness Score Correlation

We plot the datapoints and show a regression line in Figure 11.3. Most points are captured by the regression line. For our correlation coefficients we found a Pearson's correlation a score of -0.71 and a Spearman's Correlation a score of -0.7. Both metrics indicate strong inverse correlation. Limb Length Error is a good indicator for the naturalness of animation.

	Covariance			Pearson			Spearman		
	sim	slide	nat	sim	slide	nat	sim	slide	nat
FSE	-1.06E+03	-1.77E+02	1.11E+03	-0.46	-0.15	-0.51	-0.60	0.00	-0.60
LLE	-7.52E+08	-4.02E+02	-7.06E+02	-0.73	**-0.75**	-0.71	-0.70	**-0.60**	-0.70
BME	-3.25E+03	-1.02E+03	-3.00E+03	**-0.92**	-0.56	**-0.89**	**-0.80**	-0.40	**-0.80**

Table 11.4: Overview of all metrics correlations

In Table 11.4 we have an overview of all correlation coefficients for all pairings of metric and human evaluation score. The FSE, LLE, BME are all taken over the first 20 seconds of animation. We compare to the Similarity Score (sim), Foot Slide Score (slide) and Naturalness Score (nat). Highlighted are the highest absolute coefficients. We did not include the Total Body Energy for our correlation overview since in its current state it produces a pattern and is not captured in a single value. BME was the best performing metric for two out of three human scores. LLE had high correlation to all human scores. The FSE was the worst performing metric for the foot slide score. Which is unexpected since it directly quantifies the foot slide of the animation.

Chapter 12

Conclusion

12.1 Conclusion

Our double perspective Kinect setup allows for potentially more accurate recordings. However a trade-off is that the observed area is smaller than what a single Kinect would observe since the views need to overlap. This limits our choice in movements. We introduced our own naturalness metrics based on limb length, foot sliding and body matching. Our metrics were correlated to scores gathered from a human study. Our best performing metrics were Body Matching Error and Limb Length Error.

12.2 Future Work

The project had some shortcomings in various forms. We originally took on the challenge of using the Kinect as source of our motion data. We maximized the quality of the recording by using the dual Kinect setup.

The quality of the motion capture data is not on the level of conventional studio environments such as IR marker tracking. The importance of high quality training data for any neural net is such that we suspect a better setup would benefit the results greatly Our naturalness metrics were evaluated using subjective scores obtained via a questionnaire. Each movement resulted in one datapoint for our tests. The total amount of movements was five. This is perhaps too low an amount to draw a good conclusion. An interesting extension of this thesis would be to take intentionally worse performing versions of our nets and produce subpar animations. We let these be evaluated by humans as well. This would significantly increase the size of the questionnaire and would most likely be better deployed on web services such as Amazon Mechanical Turk. A standalone foot contact classifier would also be a good addition to the project to more reliably evaluate the foot contact when playing the animation. As a last point we would want to show the strength of taking this approach of learning motion capture data. We would show it running in an interactive environment. Would the animation be able to recover when interrupted by external forces. Also for evaluating naturalness a more lifelike avatar with a mesh would improve the evaluation process for the participants of the questionnaire.

Bibliography

[ACZ+13] Stylianos Asteriadis, Anargyros Chatzitofis, Dimitrios Zarpalas, Dimitrios S. Alexiadis, and Petros Daras. Estimating human motion from multiple kinect sensors. In *Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications*, MIRAGE '13, New York, NY, USA, 2013. Association for Computing Machinery.

[AF02] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, jul 2002.

[Aga18] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018.

[AHK18] Ryo Arima, Yoshiki Higo, and Shinji Kusumoto. Toward refactoring evaluation with code naturalness. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, page 316–319, New York, NY, USA, 2018. Association for Computing Machinery.

[BLCW09] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[BRS+11] Kai Berger, Kai Ruhl, Yannic Schroeder, Christian Bruemmer, Alexander Scholz, and Marcus Magnor. Markerless motion capture using multiple color-depth sensors. pages 317–324, 01 2011.

[DCH+16] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981.

[FLM15] Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. Recurrent network models for kinematic tracking. *CoRR*, abs/1508.00271, 2015.

[GCAA13] Avik Ghose, Kingshuk Chakravarty, Amit Kumar Agrawal, and Nasim Ahmed. Unobtrusive indoor surveillance of patients at home using multiple kinect sensors. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, New York, NY, USA, 2013. Association for Computing Machinery.

[GUGK17] Krishnam Gupta, Sarthak Upadhyay, Vineet Gandhi, and K. Madhav Krishna. Small obstacle detection using stereo vision for autonomous ground vehicle. In *Proceedings of the Advances in Robotics*, AIR '17, New York, NY, USA, 2017. Association for Computing Machinery.

[HKS17] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Trans. Graph.*, 36(4), July 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[HSKJ15] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, SA '15, New York, NY, USA, 2015. Association for Computing Machinery.

[JZW+15] Feng Jiang, Shengping Zhang, Shen Wu, Yang Gao, and Debin Zhao. Multi-layered gesture recognition with kinect. *J. Mach. Learn. Res.*, 16(1):227–254, jan 2015.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, jul 2002.

[Kir08] Wilhelm Kirch, editor. *Pearson's Correlation Coefficient*, pages 1090–1091. Springer Netherlands, Dordrecht, 2008.

[KL21] Gyoo-Chul Kang and Yoonsang Lee. Finite state machine-based motion-free learning of biped walking. *IEEE Access*, PP:1–1, 01 2021.

[LCR+02] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, page 491–500, New York, NY, USA, 2002. Association for Computing Machinery.

[Li12] Yi Li. Multi-scenario gesture recognition using kinect. In *Proceedings of the 2012 17th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational amp; Serious Games (CGAMES)*, CGAMES '12, page 126–130, USA, 2012. IEEE Computer Society.

[LYLL22] Xia Liu, Mingdie Yan, Yazhuo Li, and Xiao Li. Evaluation system of push-up action based on kinect. In *2022 14th International Conference on Machine Learning and Computing (ICMLC)*, ICMLC 2022, page 308–312, New York, NY, USA, 2022. Association for Computing Machinery.

[MJL18] Bill Manaris, Leslie Jones, and Erin Leigh. Liminalspace: for cello, motion capture, and interactive software, 07 2018.

[MYYL19] Hao Ma, Weichao Yan, Zaiyue Yang, and Haoyang Liu. Real-time foot-ground contact detection for inertial motion capture based on an adaptive weighted naive bayes model. *IEEE Access*, PP:1–1, 09 2019.

[NSN13] Niels Christian Nilsson, Stefania Serafin, and Rolf Nordahl. The perceived naturalness of virtual locomotion methods devoid of explicit leg movements. In *Proceedings of Motion on Games*, MIG '13, page 155–164, New York, NY, USA, 2013. Association for Computing Machinery.

[PSJ+11] J. S. Prasad, Advitiya Saxena, Nilesh Javar, K. B. Kaushik, P. Chakraborty, and G. C. Nandi. Gesture recognition by stereo vision. In *Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia*, IITM '10, page 155–162, New York, NY, USA, 2011. Association for Computing Machinery.

[RGRM14] Simon Ramm, Joseph Giacomin, Duncan Robertson, and Alessio Malizia. A first approach to understanding and measuring naturalness in driver-car interaction. In *Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '14, page 1–10, New York, NY, USA, 2014. Association for Computing Machinery.

[RMYZ11] Zhou Ren, Jingjing Meng, Junsong Yuan, and Zhengyou Zhang. Robust hand gesture recognition with kinect sensor. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, page 759–760, New York, NY, USA, 2011. Association for Computing Machinery.

[RPE+05] Liu Ren, Alton Patrick, Alexei A. Efros, Jessica K. Hodgins, and James M. Rehg. A data-driven approach to quantifying natural human motion. *ACM Trans. Graph.*, 24(3):1090–1097, jul 2005.

[SMK19] Kyle Stewart, Bill Manaris, and Tobias Kohn. K-multiscope: Combining multiple kinect sensors into a common 3d coordinate system. In *Proceedings of the 6th International Conference on Movement and Computing*, MOCO '19, New York, NY, USA, 2019. Association for Computing Machinery.

[spe08] *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, New York, NY, 2008.

[ULB04] Adrian Ulges, Christoph H. Lampert, and Thomas Breuel. Document capture using stereo vision. In *Proceedings of the 2004 ACM Symposium on Document Engineering*, DocEng '04, page 198–200, New York, NY, USA, 2004. Association for Computing Machinery.

[vdW20] Ruben van der Waal. Biped motion synthesis using kinect 2 motion capture data and phase functioned neural networks. "Leiden University Research Project Report", LIACS, Leiden University, 03 2020.

[YLP07] Kangkang Yin, Kevin Loken, and Michiel Panne. Simbicon: simple biped locomotion control. *ACM Trans. Graph.*, 26:105, 07 2007.

[YTL18] Wenhao Yu, Greg Turk, and C. Karen Liu. Learning symmetric and low-energy locomotion. *ACM Trans. Graph.*, 37(4), jul 2018.