Smoothing techniques for 3D animated human pose estimation data

August Bergöö, Oliver Lundgren

Master's thesis
2024:E48

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

Virtual 3D reconstructions of live sport events are on the horizon and to produce a high quality experience for viewers it is important that the movements of the 3D models look natural. Today, state of the art pose estimators produces data that contains noisy data, resulting in jittery animations with pose errors. The goals for this thesis is to evaluate the performance on an LSTM and classical filters in their ability to reduce such noise, and if they can be used to improve the viewer experience. This was done by comparing the Mean Per Joint Positional Error (MPJPE) and Absolute Accerational Error (AAE) for artificially added noise to motion capture data and comparing the results. A qualitative survey was also constructed to evaluate how the different methods affected the human perception of the animations when focusing on different aspects. We concluded that our methods were (to varying degrees) able to reduce the amount of noise introduced to the motion data. The qualitative study also indicated that the Savitzky-Golay smoothing algorithm improved the participants perception of the animations.

Acknowledgements

Before we proceed, we want to thank our supervisors at Spiideo, without the guidance and assistance from Håkan and Erik the overall quality and standard of this thesis would not have been the same. Additionally, we would like to thank Magnus for taking the responsibility as our supervisor from LTH. Great appreciation also goes to the other thesis workers at Spiideo for always being available to discuss ideas, solutions, and improving the quality of life during the entire semester. Everyone's support has been invaluable. Lastly, thank you to the employees of Spiideo for making us feel at home in a great office environment.

Contents

1. Introduction **9**
- 1.1 Problem formulation . . . 9
- 1.2 Research Questions . . . 9
- 1.3 Related Work . . . 10
- 1.4 Scientific Contribution . . . 10
- 1.5 Work Distribution . . . 10

2. Theory **11**
- 2.1 3D character animation . . . 11
- 2.2 Rotational representations . . . 13
- 2.3 Classical filters . . . 14
- 2.4 Machine Learning . . . 15
- 2.5 Metrics . . . 19

3. Method **20**
- 3.1 Setup . . . 20
- 3.2 Noise modelling . . . 21
- 3.3 Filter Implementation . . . 23
- 3.4 LSTM implementation . . . 26
- 3.5 Qualitative Study . . . 28

4. Results **30**
- 4.1 Filters . . . 31
- 4.2 LSTM models . . . 35
- 4.3 Combined results . . . 37
- 4.4 Qualitative study . . . 39

5. Discussion **40**
- 5.1 Findings . . . 40
- 5.2 Method Analysis . . . 41

6. Conclusion **47**

Bibliography **48**

1 Introduction

With an increased interest across many industries for VR and AR applications, the subject of 3D animation is getting more relevant. When animating a character there are lots of things that can go wrong in the pursuit of making the characters movements feel natural and realistic. This is especially true in the case of 3D sports streaming, where athletes movements are critical to the experience. When creating a 3D representation of a sports experience such as soccer, the poses of the players need to be continually estimated and updated. These estimations are then used for representing the animated movements of the 3D representations of the players.

Often these estimations of human poses contain noise, either in the form of small inaccuracies that result in jittering character models, or errors in the estimation which lead to limbs of the player models being represented poorly. The latter can cause phenomena which make the experience feel unrealistic, such as feet sliding on the floor or arms clipping through the body in unnatural ways.

The focus of this work will be to use different techniques to mitigate the effects of this noise.

1.1 Problem formulation

This thesis will compare and evaluate the performance of different methods to process noisy animation data for human characters. Both classical filtering methods and neural network models will be applied and evaluated. To evaluate the results a combination of quantitative and qualitative metrics will be used.

1.2 Research Questions

Going forward with the method and design choices of this thesis, finding answers to the following questions will be considered:

- How does an LSTM Neural Network compare to classical filters in performance when reducing jitter and pose errors in 3D pose estimation data?
- Can these methods be used to improve the user experience?

1.3 Related Work

In [Shao et al., 2023], some neural network models for predicting and smoothing human motion data are constructed and evaluated using metrics similar to the ones used in this thesis. In [Graßhof et al., 2023], a Neural network based solution for smoothing estimated noisy human motion data is presented, along with a noise model that emulates pose estimation data. In [Memar Ardestani and Yan, 2022] B-Spline filtering is used to reduce noise in motion capture data.

1.4 Scientific Contribution

This thesis proposes and evaluates the performance of different filtering approaches along with a LSTM neural network solution for smoothing noisy human motion data. A model for emulating noise that is associated with existing pose estimation solution is proposed and used as a baseline for the tested smoothing methods. A qualitative study is conducted to measure human perception of motion data with different properties.

1.5 Work Distribution

Author	August (%)	Oliver (%)
Report Writing	50	50
3D environment	20	80
Noise modelling	100	0
Neural Networks models	95	5
Filters	10	90
Qualitative study	50	50

2 Theory

2.1 3D character animation

In most cases when animating a character in a 3D environment an underlying structure of joints are used for morphing the character model. This results in being able to pose the parts of the 3D model in different ways, without the need for generating an entirely new 3D model. [Song et al., 2015]

These structures of joints, usually called *skeletons* are represented as nodes mapped to joint positions of the 3D model, with a hierarchy to represent how some joints are connected. For a 3D character this could be exemplified by a node representing the elbow having a parent node that represents the shoulder.

Each node contains data that represents its position relative its parent joint, through positional or rotational coordinate systems. To achieve motion, a time series of coordinates mapped to each node is applied to the skeleton. Each set of coordinates in the time series represents the pose of the character at one time instant. These sets, henceforth referred to as *frames*, are iterated through to animate the skeleton, which in turn makes the 3D model move.

When a translation or rotation is applied to a joint the children of the corresponding node will inherit this attribute and apply its own local translation or rotation to calculate the global position and direction. These global coordinates can then be used to draw the model in context to the rest of the scene.

2.1.1 The SMPL model

SMPL is a model proposed by [Loper et al., 2015] consisting of a parametrisation of the human body. It contains parameters for body shapes and poses to accurately represent human motion in a 3D space. For this thesis, the most interesting aspect of the SMPL model is the parameters representing the pose, since it is the estimation of the pose that we aim to improve.

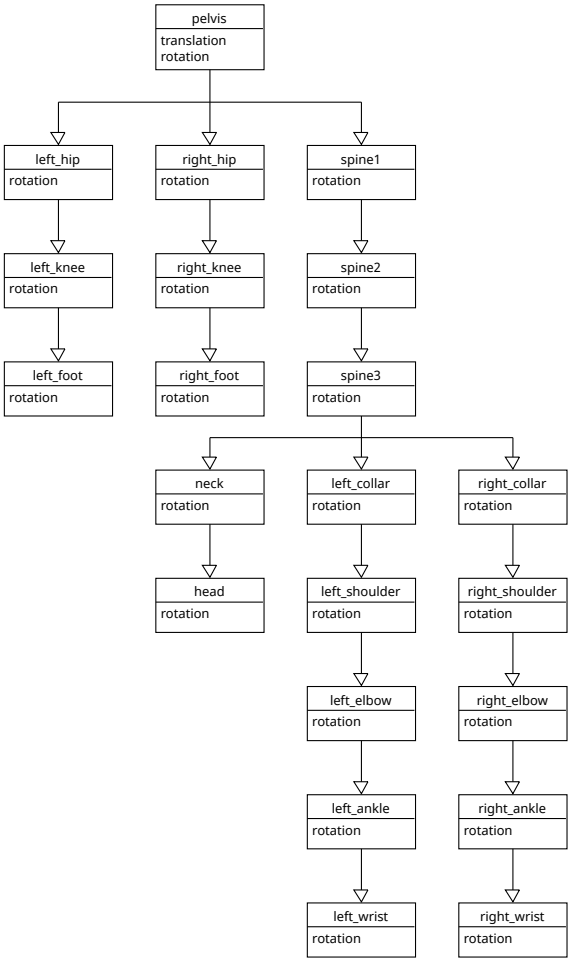


Figure 2.1 The joint hierarchy of the SMPL model. Note: `right_hand` and `left_hand` are missing from this graph, since we ended up not using those in our work. This leaves 22 joints.

Skeleton structure

To represent a pose using the SMPL model, 24 nodes are used, the hierarchy of which can be seen in 2.1. To represent movement, new coordinates representing rotations and translations are applied at each frame. The translation component of `pelvis` is used as the global translation of the skeleton. This means that if the animation includes a movement where the entire 3D model needs to move (e.g. walking forwards) only the translation component of `pelvis` needs to be updated to translate the entire model in a certain direction, since the positions of all other joints inherit their global translation from this one.

The animation data consists of multiple frames containing local rotations for each of the joints, along with the global translation of `pelvis`.

2.2 Rotational representations

Rotations in three dimensions can be described in multiple ways. Three degrees of freedom are required to fully describe an orientation in space, though, in the world of animation this may result in complications that will be described below.

2.2.1 Rodrigues' formulation

Axis-Angle Representations is a way to describe a rotation using a unit vector, $\hat{\boldsymbol{e}}$, which describes the axis of rotation and an angle, θ, which denotes the magnitude of the rotation [Curtright et al., 2014]. The rotation, $\boldsymbol{r}$, can then be described with

$$\boldsymbol{r} = \theta\hat{\boldsymbol{e}} \tag{2.1}$$

The Rodrigues vector is encoded with a different scale on the rotation angle, based on the tangent function. Here, the rotation is calculated by

$$\boldsymbol{g} = \tan\frac{\theta}{2}\hat{\boldsymbol{e}} \tag{2.2}$$

Using Rodrigues parameters introduces new characteristics, for example, g is now a discontinuous function as a result of using tangent. The dataset used in this thesis have rotations described with Rodrigues formulations.

2.2.2 Quaternions

Another method for encoding Axis-Angle rotations is with quaternions. Quaternions are 4-dimensional vectors which describes rotations in three dimensions, introducing to some distinctive characteristics. Notably, quaternions enables the ability to describe the same rotation in multiple ways. The quaternion q is the same rotation as $-q$ [Dam et al., 1998]. This property, coupled with a concept called *unrolling*, can effectively mitigate the discontinuities inherent in using Rodrigues formula.

2.2.3 Unrolling

For quaternioins the issue of discontinuity can still be present if not properly preprocessed, this is because of the ability to describe the same rotation with either $\pm q$. The process of unrolling involves examining and 'correcting' the data by adjusting the polarity of the quaternion q to ensure it remains on the hemisphere closest to the preceding rotation, thus maintaining continuity and coherence in the rotational data [Holding, 2024].

2.3 Classical filters

2.3.1 B-splines

A *spline* is a mathematical definition of a group of functions. These functions are piecewise polynomials built up of several smaller intervals which are based on control points or knots. Depending on the type of spline the segments have a varying degree of continuity. Since one of the goals of this project is to reduce noise and jitter and introduce a smooth and lifelike feeling to animation data, at least a second degree of continuity is needed, meaning that the second derivative is continuous for all points on the curve [Ibrahim et al., 2017]. For these applications there is a subgroup of splines called basis splines, often known as *B-Splines*. These have characteristics that make them viable for data smoothing and noise reduction. One of these features is the ability to have local control, improving results if there are segments more susceptible to noise than others by allowing varying smoothing rates. Which degree of continuity is also adjustable, further improving noise reduction capabilities when it is correctly adjusted by application purpose. [De Boor, 1976]

2.3.2 Savitzky-Golay filter

Another, more simple filtering method is the Savitzky-Golay filter, also known as a *Savgol filter*. The concept of the algorithm is using the least squares method to find a curve of a specified degree based on a predetermined number of points [Liu et al., 2016]. Consider a choice of b points, creating a window between $x_0, .., x_b$. The algorithm fits a polynomial to these b data points. To choose the value of this window for the smoothed result, F_t, at time $t = \frac{b}{2}$. The next estimation is then based on the same principles but for the window of input data $x_1, .., x_{b+1}$. Looping though these steps until estimations are calculated for all data points results in a smoothed estimation of the original data [Mishra et al., 2019]. Careful consideration of the window size and polynomial degree is essential to prevent overfitting and excessive smoothing that might obscure relevant details amidst the noise. This filter was chosen as it is a frequently used algorithm for detecting and improving data with noise.

2.3.3 Double Exponential Smoothing

A popular technique, used in time series analysis, financial forecasting and noise reduction etc. called *Double Exponential Smoothing* (DES) is valued for it's efficiency and real-time attributes when compared to other methods and algorithms used for similar purposes [Chung and Kim, 2013]. This algorithm has two components, a level and trend at time/frame t, denoted a_t and b_t respectively. These are calculated with

$$\begin{aligned} a_t &= \alpha x_t + (1-\alpha)(a_{t-1} + b_{t-1}) \\ b_t &= \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1} \end{aligned} \tag{2.3}$$

where α is the smoothing constant for level and β for trend. Using the above equations to construct the formula responsible for estimating the values on frames

$t+h$ as follows;

$$F_{t+h} = a_t + hb_t \tag{2.4}$$

where h is amount of frames ahead. This algorithm is included in the thesis because of the potential smoothing properties as well as the option to use it as a position forecaster potentially allowing increased real-time applications.

2.4 Machine Learning

One area of research that has gotten a massive amount of attention in the last few years is machine learning. Machine learning is a part of the realm of Artificial Intelligence where machines learn from data, with the ultimate goal of finding patterns to generalize and make reasonable assumptions about previously unseen data.

In general a machine learning model tries to approximate a function based on some inputs and outputs. One example could be a classifier where the inputs are images and the outputs are the types of objects in the images. In cases where the corresponding true output values are known these are referred to as *ground truth*.

2.4.1 Loss function

When training a machine learning model, a *loss function* is used to evaluate the performance of the current model. The model and its parameters are then improved upon with respect to the loss function to hopefully approximate the ground truth data in its output more accurately.

The objective function is formulated using a way to quantify the error of each input-output pair in a model. It could be the mean squared error distance between the output and the ground truth, or a more complicated relationship between them.

2.4.2 Artificial Neural Networks

Artificial Neural Networks are models inspired by biological neural networks such as the human brain. One basic and common type of Neural Network, the feedforward neural network, consists of a number of layers of connected Artificial neurons. Information flows from the input layer through all other layers up until the output layer, which will approximate some function of the inputs, or *features*. In each layer are units, often called neurons, representing an *activation function* of the inputs from the preceding layer and outputting the function value to the next layer. At each step the output of each node is weighted, biased, and then fed into the activation function of the next layer in the network. The weights and biases of each connection are the parameters that are optimised when training a neural network. [Goodfellow et al., 2016]

Training

When training a neural network, the weights and biases of the network are updated according to the examples of input-output pairs of the model according to the objective function and some optimiser. The process most commonly used is called *stochastic gradient descent*. The procedure can loosely be described as follows: [Goodfellow et al., 2016]

1. Pick a small number of examples(feature - ground truth pairs) from the training set

2. Estimate the gradient of the objective function based on those examples, with respect to the model parameters(the weights and biases)

3. Update the model parameters in the "direction" of the steepest descent according to the estimated gradient, using some step size, often referred to as *learning rate*

4. repeat until a local minimum of the objective function is reached

Recurrent Neural Networks

In contrast to feedforward neural networks, recurrent neural networks(RNNs) allow for feedback connections between the layers of the network. This is useful in time series predictions, as it allows for neural networks that can process input sequences of arbitrary length in the time dimension, as each layer can apply the same function to each time step and propagate the result forward.

It does however present one problem, namely that of the *exploding/vanishing gradient* problem. This refers to a problem that can occur when calculating the gradients for optimising on a RNN, where the propagation of a gradient over the many stages has a tendency to disappear or explode. This is because of the exponential nature of applying the same function to each input step and propagating forward. The results of this problem can either be an "explosion" of the gradient where the values propagated through the time stages get larger with each iteration, or more commonly the values get exponentially smaller with each iteration and the information from time steps further back are rendered insignificant compared to the values of recent inputs. [Goodfellow et al., 2016]

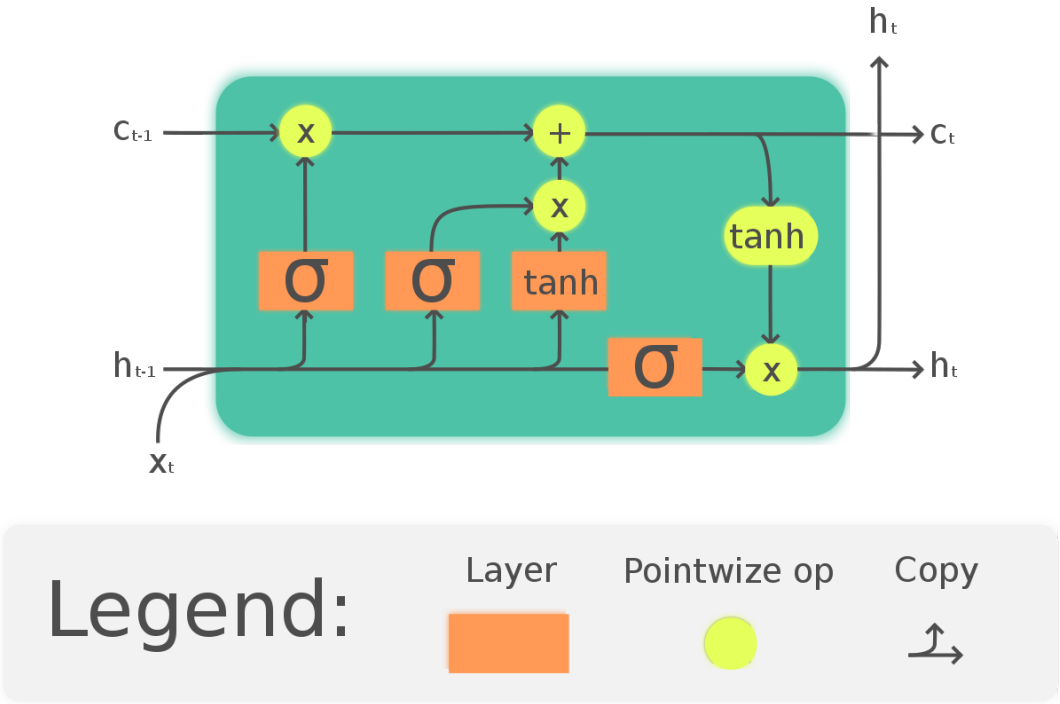


Figure 2.2 The structure of a basic LSTM cell. σ refers to the activation function of the layer being a sigmoid function. c and h are the inputs/outputs from the previous/to the next layer, and x refers to the input for the current time step. Image taken from [Chevalier, 2021]

2.4.3 Long Short-Term Memory networks

One solution that has been proposed that addresses the vanishing gradient problem comes in the form of *Long Short-Term Memory networks* (LSTMs). The model was first proposed in [Hochreiter and Schmidhuber, 1997]. The most commonly used LSTM models use a *gated* architecture and a *memory cell*, as is shown in fig. 2.2. The model builds on the same principle as a RNN, with a distinction being that every cell has the ability to store information in a separate channel, the cell state. The cell state (c_t in the figure) is modified through gates consisting of *sigmoid* layers, which give as output a number between 0 and 1, which is to be interpreted as the fraction of information to keep for the output to the next cell and time step. These gates are used for conditioning both the cell input (both from the previous cell and the current time step) and the output. Because of this, the exponential behaviour usually seen in RNNs is no longer present. [Lipton et al., 2015]

2.4.4 ML in practice

In designing a machine learning model the variables defining the general training behaviour or configuration of the model are called *hyperparameters*. These hyperparameters can be tuned to make the model learn from the data as effectively as possible. Finding the best set of hyperparameters for a particular problem is called hyperparameter optimisation, and is commonly used when defining ML models.[Feurer and Hutter, 2019]

Data partitioning is a fundamental concept in machine learning. It involves dividing a dataset into distinct subsets to create different stages of model development and evaluation. The primary objective is to ensure integrity of model evaluations. By separating into train, validation and test sets the performance of models can be assessed in a systematic way to increase integrity and reduce the risk of not detecting *overfitting*.

Overfitting can occur when a model memorizes the input instead of learning its patters. Instead, the training set is used to train the parameters of the models, where it learns the underlying patters of the data. The validation set, kept separate from the data that the model accesses during training, is then used to assess how well these parameters perform. The validation performance can then be used as insight into how well the model performs on unseen data, which is important for not overfitting the training data, as well as optimising the hyperparameters.

Finally, the test set serves as an unbiased assessment of the model's performance on completely unseen data, to provide insights into its real-world applicability. This test set result can not be used to change the model, as the model then becomes biased and the test set can no longer be counted as unseen. [Muraina, 2022]

2.5 Metrics

There needs to be a way of evaluating how the methods described in the sections above perform. Two common methods are *Mean Per Joint Position Error (MPJPE)* and *Absolute Acceleration Error* [Wang et al., 2021][Shao et al., 2023].

2.5.1 Mean Per Joint Position Error

This metric quantifies the Euclidean distance between the ground truth position of joints and their predicted coordinates. Since the dataset for this thesis is from motion capture, the recorded data can be assumed to be ground truth. For each 3D node, the difference in position between the estimations and ground truth for each frame gets accumulated. This value is then divided by the total amount of joints to get the MPJPE. The lower value the better.

2.5.2 Absolute Acceleration Error

Absolute acceleration error, AAE, is defined in a similar way to MPJPE, using the estimated and ground truth 3D node positions. To estimate the acceleration a of a node, its position p_t along with the previous position, p_{t-1} and the next position p_{t+1} in time are needed. Since acceleration is the change of velocity, we can estimate it with the velocity going to the next position minus the velocity in coming from the previous one:

$$a = \frac{p_{t+1} - p_t}{\Delta t} - \frac{p_t - p_{t-1}}{\Delta t}) = \frac{p_{t+1} - 2 * p_t + p_{t-1}}{\Delta t} \tag{2.5}$$

Where Δt is the time passed between frames(here assumed to be constant). AAE is then calculated as the distance between the acceleration a for the estimated and the ground truth nodes.

3 Method

3.1 Setup

3.1.1 Bevy

In this project we used an open source game engine called BEVY. This was done as in addition of being open source, the engine offers beneficial features in terms of efficiency. One of these features is that the engine is fully modular and plugin based allowing the user to specify which parts of the engine to compile and run. This allowed us to create two versions of our main program. One to visualize the results and the other to run calculations where we could exclude the part of the engine responsible for showing graphics and audio. This also gave the opportunity to use user created plugins, which could automatically introduce lights and movement controls in the environment saving us energy and time. Another feature, and the driving factor for this choice of engine is the fact that BEVY is Entity Component based.

Entity Component Systems (ECS) is the backbone of BEVY and as stated above one of the main reason for Spiideos interest in the engine. This pattern is based on creating entities that contain components. instead of typical object oriented programming. For this project each player is an entity consisting of the 3D model as well as a *player pose* component containing information about that specific players rotation. Moving the characters is then as simple as creating a query requesting every entity with this component. By iterating though this query the positions can be updated for every individual player.

It is worth noting that using BEVY is a choice that will result in a lot of challenges, especially if there is no previous experience with ECS or rust. BEVY is still in development and changes a lot between different versions. A lot of the information online was outdated. How BEVY effected this project will be further evaluated in the discussion.

3.1.2 Datasets

The experiments conducted during this thesis were done and evaluated on the *BEDLAM* dataset, presented in [Black et al., 2023]. This dataset contains, among other data, animation files for over 10 000 human motion sequences. Each animation consists of around five seconds of motion recorded at 30 fps.

Each animation is stored inside a *numpy*-array file(`.npz`), consisting of frame-wise translation and joint rotation data in Rodrigues formulation as described in section 2.1. The animations all contain rotational data for all joints in the slightly expanded *SMPL-X* model, which contains separate nodes for facial features and individual fingers(totaling 55 joints). These extra joints were deemed uninteresting for the purposes of this thesis and were therefore cut from the testing to reduce the scope of our models, and so the final model used 22 joints (as shown in 2.1).

3.2 Noise modelling

Because the BEDLAM dataset consists of motion capture data, the errors and noise present in 3D pose estimation are insignificant. This means that the dataset could be used as ground truth, however it also presented the need for a way to realistically simulate these artefacts and apply them to the dataset to find an estimation for our input data.

Two models for noise were developed for this thesis. The first model was heavily inspired by the noise presented in [Graßhof et al., 2023], while the second model expanded on some features to further similarities to real world 3D pose estimation data. The models are explained in detail below.

3.2.1 Initial noise model

This model is based on two parts, imprecision noise (*jitter*), and corrupted frames. The noise N is added to the rotational data r of each joint in every frame, creating the noisy data r'. This process is described below:

$$r'_{i,j,k} = r_{i,j,k} + N_{i,j,k} \tag{3.1}$$

Where i is the frame, j is the joint and k is the coordinate of the rotational data. The noise function N is sampled at each coordinate k as described below:

$$N_{i,j,k} \sim \mathbf{N}(0, \tilde{\theta}_i) + \mathbf{B}(p_{i,j}) \cdot \mathbf{N}(0, \hat{\theta}_i) \tag{3.2}$$

Where $\mathbf{N}$ and $\mathbf{B}$ denote a Gaussian distribution and a Bernoulli distribution, respectively. $p_{i,j}$ is the probability for the current frame being corrupt. $\tilde{\theta}_i$ and $\hat{\theta}_i$ are each sampled once every animation, which means they are dependent on frame i. They are sampled from distributions shown below, where μ_B, μ_L are the mean values of the distributions and θ_B, θ_L are their standard deviations.

$$\tilde{\theta}_i \sim \mathbf{N}(\mu_B, \theta_B) \tag{3.3}$$

$$\hat{\theta}_i \sim \mathbf{N}(\mu_L, \theta_L) \tag{3.4}$$

We observed in some applications, such as [Bridgeman, 2019], that missing frames for pose estimation often occurred in a series close to each other. This property was simulated by varying the parameter the probability of a frame being corrupt. A base probability was assigned to it, with a large increase if the previous frame was also corrupt. This made longer sequences of corrupt frames more probable than single outliers, leading to the desired behaviour.

3.2.2 Final noise model

The previous model was iterated upon and improved to more accurately depict the artifacts described in 1.1. The focus was shifted from missing and corrupted frames toward estimation errors where the limbs of the model do not match those of the real pose. The approach to simulate these erroneous predictions was to exaggerate or dampen the motions in the animations randomly. This lead to body parts clipping into one another and other impossible human poses being added to the noisy input data. The process of adding the noise this time around is described in 3.5

$$r'_{i,j,k} = \alpha_{i,j} * \beta_{i,j} * r_{i,j,k} + (1 - \alpha_{i,j}) * r_{i,j,k} + N_{i,j,k} \tag{3.5}$$

In this case, α controls whether a faulty joint estimation is occurring, in which case an exaggeration or dampening is sampled from β, which skews the rotational coordinates and creates distorted movements. N represents the jitter as described before. N is sampled for every joint coordinate, α for each frame and β for every joint from the distributions described in 3.8.

$$N_{i,j,k} \sim \mathbf{N}(0, \tilde{\theta}_i) \tag{3.6}$$

$$\beta_{i,j} \sim \mathbf{N}(1, \theta_O) \tag{3.7}$$

$$\alpha_{i,j} \sim \mathbf{B}(p_{i,j}) \tag{3.8}$$

$\tilde{\theta}_i$ sampled each animation sequence as described in eq. 3.3. The parameter $p_{i,j}$ is modelled to reflect the serial nature of estimation errors, meaning that an erroneous limb was often followed up with a series of frames where that same limb had its coordinates distorted by the same amount, β. This process was simulated in the same way as the corrupted frames were implemented for the initial noise model. A mechanism was also put in place to make sure that β is only sampled once for each sequence of simulated estimation errors. The reason is that the entire sequence would thus be distorted in the same way, resulting in a fluid but incorrect movement.

3.2.3 Parameter choices

The effects of the parameter values $\mu_B, \mu_L, \theta_B, \theta_L, \theta_O, p$ for the distributions were visually inspected when applied to an animation and tweaked until a reasonable amount of noise was achieved, and the animations looked roughly like the animated pose estimations described in 1.1. These choices were:

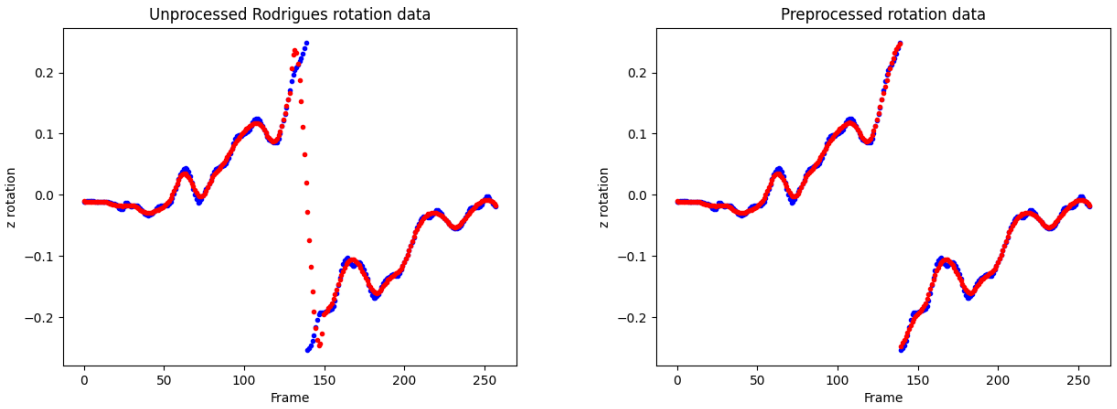
μ_B	0.01
μ_L	1
θ_L	0.002
θ_B	0.01
θ_O	0.3

The probability for a frame being corrupted in the first noise model was chosen to be 0.02 in the baseline case, increasing to 0.75 if the previous frame was corrupted.

In the final noise model the baseline probability for estimation error was 0.03, increasing to 0.96 if the limb had been distorted in the previous frame.

3.3 Filter Implementation

As mentioned in the theory, describing rotations for three dimensions can result in discontinuities in the data. By transposing these 3D vectors to quaternions and unrolling them we can prevent this causing issues during filtration. This can be seen in fig. 3.1, where (a) shows in red how the data is smoothed before quaternion transformation and unrolling, compared to (b) where this is implemented. The filtration algorithms used in this thesis are all one dimensional. This means that to use these methods, the data must undergo parsing from quaternion containing x, y, z and w rotations to one-dimensional scalars representing rotations within a single channel. This was done by iterating though the data, identifying the different channel rotations for each joint and saving them their own array. This transition enables the description of body part movements from multi-dimensional vectors for each frame to a unified channel.



(a) Rotation in z channel without any preprocessing. Red dots shows the smoothing predictions on the blue input data.

(b) Rotation in z channel after the data has been transformed to quaternions, unrolled, smoothed and transformed back to Rodrigues form. Red dots show the smoothing predictions on the blue input data

Figure 3.1

B-Spline

As this is a well established method in the world of computer science the algorithm is available in the python library called *Scipy* [Virtanen et al., 2020]. This algorithm has three required inputs; t, c and k where t is the array of knots based on the data points, c the spline coefficients, and the B-spline degree k. Using another Scipy function, *splrep*, these parameters can be calculated from the data points. Besides the inputs x and y, which denote the frames and node rotations within a single channel respectively there is a set of non-required parameters that can be adjusted for specific purposes. In this thesis we use the standard values for all but one of these parameters, namely; the smoothing condition s. As the name implies this adjusts the amount of smoothness, with higher s resulting in a higher degree of smoothing. This parameter will be chosen based on the results from the parameter analysis described in section below.

Savgol filter

Just as the method above, Savgol filters have been around for a long time and the algorithm is already implemented in a multitude of different libraries. In our case, a version is imported from Scipy. It has eight different parameters, with three of them being non-optional. These three are; the data for one channel, x, the window length, wl, and the order of the polynomial used, po. The default values will be used for the other parameters. Both window length and polynomial order will be based on the parameter analysis.

Double Exponential Smoothing

A version of this method can be found in smaller and less popular libraries but as it is a simple algorithm we decided to be sure it worked for our purpose and implemented it. Creating our python function with parameters for the one dimensional channel as well as an α and β for the level and trend. The functions calculating the smoothed data follows the equations 2.4 and 2.3. Since this is a forecasting method that takes previous values into account the first value cannot be calculated, instead this is chosen to be the same as the first value of the input data. With this information it was a simple path to implement a working version.

3.3.1 Parameter Analysis

Since the chosen algorithms above all possess adjustable parameters that impact how they work and calculate their results is is essential to have an appropriate way to evaluate which values these parameters should take. By analysing these values rather than relying it on intuition, the validity of the results increase. Therefore, a parameter analysis was conducted.

Initial Parameters

How the initial parameters are selected can have an impact on the results of the analysis. This can be especially important if the outcomes of the filters has local minimums that differ from the global. To mitigate this risk, for all individual parameters, the starting value was chosen to be lower than the reasonable interval. If this value performed even marginally well, it was lowered until the result of the filtering yielded an objectively poor result. Then, increasing the input until receiving the same poor results. To further eliminate the risk of this being a local minimum, testing past this high value is done to evaluate if the results improve. If the results do not improve, a reasonable span of initial parameters have been found.

Evaluations

In this thesis, the performance evaluation is based on MPJPE and AAE. How these errors should be weighted has an impact on the best performing parameter. By introducing the weight w, the total error can be adjusted with $e_{error\_valuation} = we_{error\_mpjpe} + (1-w)e_{error\_aae}$, enabling prioritisation of one of the errors.

Strategy 1: Grid Search

With the aid of the initial parameter search, a wide interval of test parameter spanning over the tested area was chosen. A subset of animations from the training set is chosen and for each of these animations noise is added, and then the calculations begin. Using the same noise, the filters iterate through all chosen parameters computing both MPJPE and AAE. Based on the error values calculated and the weight w, the best parameter of each filter for this animation is recorded together with the noise and outcome it produced. Once all animations in the training batch have been used, an average of the best performing parameter is calculated. For each batch, the values for; frame averaged MPJPE and AAE, using the best parameter, can be compared between filters giving a picture of which filter performs the best.

Strategy 2: Adaptive Grid Search

Similar to how *Strategy 1* operates, this approach begins with the same initial parameters. However, the difference is now that at the end of the calculations for an animation, before the next, the interval for the grid search is adjusted. The updated values are determined based on the average values of the best performing parameters during the last five animations. This gives the opportunity to test smaller intervals obtaining more precise parameter values, with the aim of focusing on parameters close to the values that consistently perform the best.

3.4 LSTM implementation

The dataset being in the form of a time series of rotational coordinates for each joint and coordinate, combined with the promised performance that LSTM models for these types of data(as described in section 2.4.3), led to the decision to try and train one for our purposes.

3.4.1 PyTorch

The implementation was done using *PyTorch*, which is a widely used open source machine learning library. PyTorch allows for easily defining and training neural network models. PyTorch has a ready implemented LSTM module, that can be used when constructing a neural network. To use it, a couple of parameters need to be defined:

- `input_size`: The number of inputs that each time step would consist of
- `hidden_size`: The dimension of the hidden state between LSTM cells(h_t in fig. 2.2)
- `num_layers`: Number of LSTM cell layers

The output from this LSTM module will be of size `hidden_size` for each time step. This dimension would not necessarily coincide with `input_size` which is the shape we want our prediction to be. We also want to map the outputs from the output space onto the target output space of the rotational coordinates. These issues are both solved by adding a layer of nodes with linear activation functions after the LSTM layers. This was implemented with a module from PyTorch, where the input size was set to the same as `hidden_size`, and output size was set to the dimension of the desired output.

3.4.2 Model design

The first decision that needed to be taken was to define what behaviour was wanted from the model. The general idea was to feed the model a number of subsequent (noisy) frames, and have the model predict what the true rotations for the last frame in the sequence would be.

Each frame of the input data contained rotational data for 22 joints in 3 dimensions. `input_size` was therefore set to 66, to capture the behaviour of the entire model at each time step. To try and find more complex patterns, `hidden_size` and `num_layers` can be set to a larger value, with an increased requirement for computational power during training and prediction. The parameters were tweaked by hand until a model could be trained within a reasonable amount of time(roughly 2 hours on a 2017 MacBook Pro), while still keeping the model as complex as possible. The final choices for these parameters were `hidden_size` = 330 and `num_layers` = 2. These were kept constant during subsequent experiments.

The model would output a sequence of frames of the same length as the one provided to it, where the last frame of the sequence would be considered the predicted frame, and compared to the ground truth when training and evaluating the model. PyTorch was used to split the dataset into batches containing sequences of frames. Since LSTM models can handle sequences of arbitrary length, the sequence length could be varied without changing the model.

3.4.3 Training and Loss

To train the model, the data set was split into a training set and an evaluation set. The evaluation set was used to evaluate the model over several training epochs. During each epoch the batches of the training set were each used to optimise the model one step, as described in 2.4.2. To calculate the gradients, the commonly used *ADAM* optimiser was used, and training was stopped when there had been no improvement for 3 epochs.

To evaluate and optimise the model, the loss function had to be defined. In early stages *mean-squared-error* between the rotational coordinates of the predicted frame and its ground truth counterpart was used as the loss function.

It was noted that the model did not perform too well on the "jitter" part of the noise, causing very shaky predictions. An attempt to mitigate this was made by alter-

ing the loss function to penalise the acceleration error of a prediction. This was done similarly to how AAE is described in section 2.5.2, where the last three predicted points of each rotational coordinate was used to calculate the equivalent acceleration value. Important to note here is that this does not automatically correspond to the acceleration error described in 2.5.2, since that refers to the acceleration of 3D points in space, and the model predicted coordinates correspond to rotations in 3D space as described in section 2.2.

The final loss function was a mix between the mean squared error loss of the predicted/ground truth coordinates and the mean squared error of the acceleration for the three last predicted/ground truth coordinates belonging to each joint.

3.4.4 Hyperparameter optimisation

The sequence length of each input and how much of the loss function consisted of acceleration error were two hyperparameters that were varied across each training run. The models were used to predict an external evaluation set(which had been kept separated from any of the data used during the training epochs). This set was then fed into BEVY, where 3D positional coordinates where recorded corresponding to the predicted frames. These coordinates were then used to evaluate the set of hyperparameters, using the metrics defined in section 2.5.

The sets which performed best using both of these metrics were both saved to be used as our final model to be compared to the other filtering methods.

3.5 Qualitative Study

Since our thesis is heavily related to the entertainment industry it is important that there is an evaluation of how the different methods are perceived, to see if these results differ from that of metric calculations. To do this a survey was created and handed out to subjects.

3.5.1 Survey Structure

The structure of the survey is important to obtaining results that can complement the objective data, strengthening or undermining the data enhancement techniques. The objective metrics are, as mentioned before, MPJPE and AAE, and how important they are to the outcome of the subjective study. To get a fair comparison the survey was divided into two segments; one where participants focused on jitter and noise and secondly one where they focused on *how natural* the animation and movements felt.

Each segment is comprised of four sets animations, consistent across both sections. Within each set, six characters are shown doing the same animation, see fig. 3.2, but their data pre-processed in different ways; one for each filtering method, one for the Neural Network, one for just the noise and finally one for ground truth. The inclusion of ground truth/noise is to evaluate if there are techniques that are pre-

ferred over the data used as the "correct value" as well as to see if there are methods that are experienced as worse than just noise without improvements.

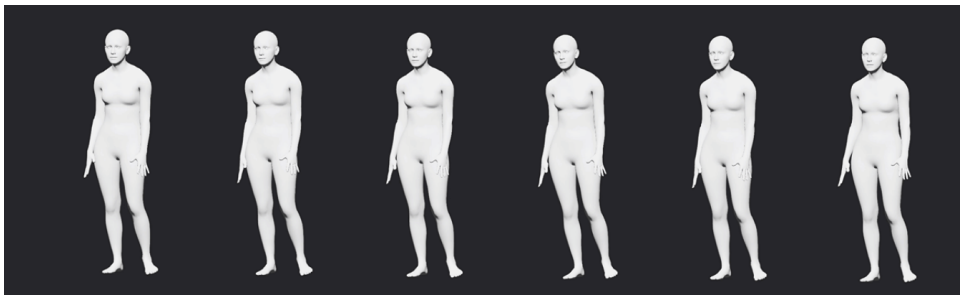


Figure 3.2 An example view of how the characters were displayed in the survey

Half of the sets were created with filter parameters and loss functions aimed at prioritizing MPJPE improvement, whilst the remaining two emphasized AAE. If the animation had MPJPE prioritization on one section it has AAE on the other, that way data is collected with both focuses for each animation.

To get an understanding of which processing method the participant preferred, two options were considered. The first option is to give the choice of rating how good they perceived the methods to be. The second option is to rank them in order of preference. Since people have different scales of how they give ratings, some may rate a characters movements as a four, but another person a 3, even though their perception of the movements are the same. To mitigate this, option two was chosen, but at the expense of not knowing if all animations perceived as terrible or good, just their ranking against each other. The order of improvement techniques used was also randomized for each set to mitigate the risk of bias resulting from repeated exposure to the animations.

3.5.2 Animation Selection

The selection of animations is also of great importance to getting results that yield meaningful conclusions. Within the BEDLAM dataset, a few animations contained noise in the ground truth data. From these, two animations with such errors were chosen. This selection aimed to assess whether our methods could improve real noise from real systems and not just noise created by our noise model. The other two animations were with normal ground truth data without any noticeable artifacts. Since one goal was to evaluate how important MPJPE and AAE are, two animations were deliberately chosen to have clear limb distortion compared to ground truth. An example can be seen in the image below, the green character displays the true position whilst the red, transparent character displays how the noise has effected the rotations. This was done to give clear examples of animations being unnatural. The other two animations chosen to have the character move around, with the intention to show true motion that has resemblance to sport movements.

4 Results

In this chapter, we present the findings of both our quantitative and qualitative studies. Beginning with the results from our parameter analysis, leading to how they are used to get the final outcomes of the filter implementations. Thereafter the results from our Neural Network will be presented, showing results from how the different weights on loss functions affected the test scores. Following this is a concise summation of all objective results. Finally, we present the findings from our qualitative study.

4.1 Filters

4.1.1 Parameter analysis

In both the standard and adaptive Grid Search the starting grid for the different methods are defined in table 4.1. Values above and below these intervals increased the error, resulting in these arrays of values being considered.

Table 4.1 Grid Search Parameters

B-Spline	**Values**
s	[0.0019, 0.0030, 0.0050, 0.012, 0.019]
Savgol	
Window Lengths	[10, 17, 23, 30, 39]
Poly Orders	[1, 3, 5, 7, 9]
Double Exponential	
Alpha	[0.1, 0.3, 0.5, 0.7, 0.9]
Beta	[0.04, 0.09, 0.14, 0.3, 0.4]

Standard Grid Search

The below table showcases the best parameters for different weights, where the displayed value is the sum of which parameter performed the best for each animation divided by the total number of animations, resulting in the average of the best performing parameters for the training set. The weight works as described in 3.3.1, where a higher w increases focus on reducing MPJPE errors, vice versa for a lower w and AAE.

Table 4.2 Each row represents one grid search, based on different weights. For each specific weight, the average of which parameter in 4.1 performed best is displayed

Weight	Best s	Best wl & po	Best α & β
0.75	0.0049	20 & 4	0.70 & 0.05
0.50	0.0056	20 & 4	0.62 & 0.06
0.25	0.0068	21 & 4	0.53 & 0.07

Where s is the parameter for B-Spline, wl and po is the window length and polynomial order for Savgol with α and β describing Double Exponential.

Adaptive Grid Search

As mentioned before, the starting grid is described in table 4.1. The showcased value is calculated just as previously as described for table 4.2. After running the training set, these were the best parameters:

Table 4.3 Each row represents one grid search, based on different weights. For each specific weight, the average of which parameter in 4.1 performed best is displayed

Weight	Best s	Best wl & po	Best α & β
0.75	0.0041	12 & 2	0.66 & 0.07
0.50	0.0044	14 & 2	0.59 & 0.08
0.25	0.0049	21 & 3	0.51 & 0.09

4.1.2 Filters

B-Spline

Using the s calculated by both the standard and adaptive grid search these are the filtration results on the validation set. The first half showcases the standard version of our grid search, the second the adaptive. The first row in each sections describe the average amount of noise per frame added to ground truth. The columns *% of Noise* indicates how much noise there is left after filtration using the specified s

Table 4.4 B-Spline validation results for **standard** using parameters from table 4.2 and for **adaptive** using parameters from table 4.3. Green cells indicates the highest performing parameters for MPJPE and AAE. Lower is better.

Standard	**MPJPE/Frame**	**% of Noise**	**AAE/Frame**	**% of Noise**
Noise	0.4413	100	0.2263	100
s = 0.0049	0.4005	90.76	0.1739	76.83
s = 0.0056	0.4028	91.28	0.1750	76.27
s = 0.0068	0.4085	92.56	0.1739	75.57
Adaptive	**MPJPE/Frame**	**% of Noise**	**AAE/Frame**	**% of Noise**
Noise	0.4413	100	0.2263	100
s = 0.0041	0.3994	90.50	0.1758	77.66
s = 0.0044	0.3996	90.55	0.1750	77.30
s = 0.0049	0.4005	90.76	0.1739	76.83

Savgol

With the window length and polynomial orders from 4.1 these are the results from the validation set. The structure of the table is the same as described for B-Splines.

Table 4.5 Savgol validation results for **standard** using parameters from table 4.2 and for **adaptive** using parameters from table 4.3.Green cells indicates the highest performing parameters for MPJPE and AAE. Lower is better.

Standard	**MPJPE/Frame**	**% of Noise**	**AAE/Frame**	**% of Noise**
Noise	0.4413	100	0.2263	100
wl = 20 po = 4	0.4072	92.26	0.1611	71.19
wl = 20 po = 4	0.4072	92.26	0.1611	71.19
wl = 21 po = 4	0.3881	87.95	0.1609	71.08
Adaptive	**MPJPE/Frame**	**% of Noise**	**AAE/Frame**	**% of Noise**
Noise	0.4413	100	0.2263	100
wl = 12 po = 2	0.4061	92.01	0.1619	71.53
wl = 14 po = 2	0.4056	91.90	0.1612	71.21
wl = 21 po = 3	0.3991	90.44	0.1600	70.70

Double Exponential

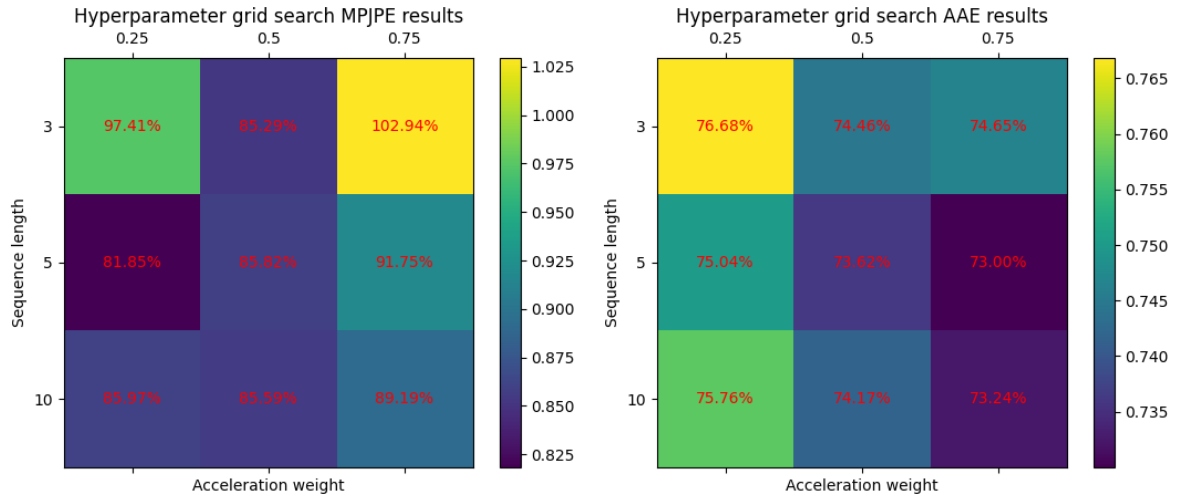
These are the results from our last filter, Double Exponential. Once again going through table 4.1 to select parameters and displaying the results using the same table structure.

Table 4.6 Double Exponential validation results for **standard** using parameters from table 4.2 and for **adaptive** using parameters from table 4.3. Green cells indicates the highest performing parameters for MPJPE and AAE. Lower is better.

Standard	**MPJPE/Frame**	**% of Noise**	**AAE/Frame**	**% of Noise**
Noise	0.4398	100	0.2975	100
$\alpha = 0.70$ $\beta = 0.05$	0.4640	105.14	0.1931	85.31
$\alpha = 0.62$ $\beta = 0.06$	0.4710	106.72	0.1861	82.25
$\alpha = 0.53$ $\beta = 0.07$	0.4860	110.02	0.1793	79.23
Adaptive	**MPJPE/Frame**	**% of Noise**	**AAE/Frame**	**% of Noise**
Noise	0.4413	100	0.2263	100
$\alpha = 0.66$ $\beta = 0.07$	0.4685	106.17	0.1901	84.00
$\alpha = 0.59$ $\beta = 0.08$	0.4762	107.91	0.1843	81.43
$\alpha = 0.51$ $\beta = 0.09$	0.4908	111.20	0.1784	78.82

4.2 LSTM models

4.2.1 Hyperparameter Optimisation



(a) MPJPE results **(b)** AAE results

Figure 4.1 Results gathered from the grid search over the different choices of hyperparameters on the external validation dataset. MPJPE metrics are shown in 4.1a, and AAE metrics in 4.1b. Lower is better.

When optimising the model hyperparameters, the results in fig. 4.1 were acquired for the different parameter choices and metrics. In fig. 4.1a the models were evaluated using MPJPE, and the best scoring model used a sequence length of 5 and an acceleration weight of 0.25. For the AAE metric, as seen in fig. 4.1b, the best scoring model used a sequence length of 5 and an acceleration weight of 0.75.

4.2.2 Training loss

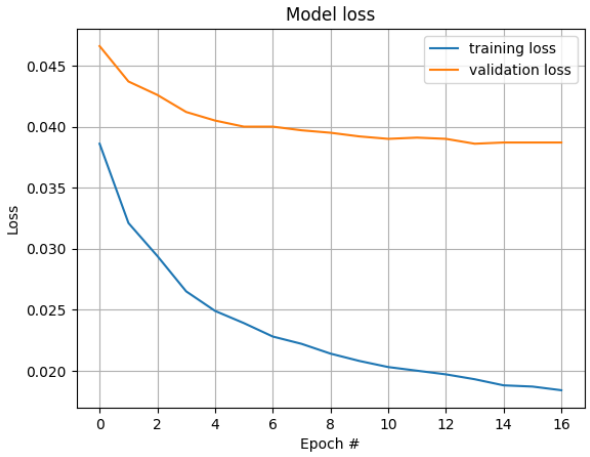


Figure 4.2 History of the loss for the training and validation sets during the training of the model that favored lower MPJPE. Lower is better.

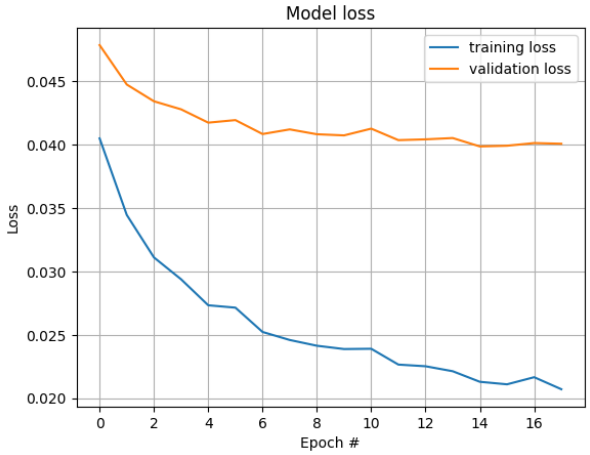


Figure 4.3 History of the loss for the training and validation sets during the training of the model that favored lower AAE. Lower is better.

The loss for the training set and the internal validation set was recorded throughout training the two models. The history for the model favoring MPJPE can be seen in fig. 4.2, and the history for the model favouring AAE in fig. 4.3.

4.3 Combined results

This section will be divided into two parts, one describing the results from the best MPJPE models and one for AAE.

MPJPE

Using the tables 4.4 - 4.6, along with fig. 4.1, the best parameters in reducing MPJPE for each method are extracted:

Table 4.7

Method	Standard/Adaptive	Parameter
B-Spline	Adaptive	$s\,val = 0.0041$
Savgol	Standard	$wl = 21$ & $po = 4$
DE	Standard	$\alpha = 0.70$ & $\beta = 0.05$
LSTM	-	seq. len. $= 5$ & acc. weight $= 0.25$

Where DE stands for Double Exponential. Results when evaluating the models with these on the test set can be seen below in table 4.8. The structure of the table is similar to before, with B-Spline having the best performance, indicated with green cells.

Table 4.8 Using the highest performing parameters for MPJPE, shown in 4.7, these results were gathered. Lower is better.

Method	MPJPE/Frame	% of Noise	AAE/Frame	% of Noise
Noise	0.4245	100	0.4432	100
B-Spline	0.3942	92.85	0.4033	91.00
Savgol	0.4001	94.25	0.3882	87.59
DE	0.4581	107.90	0.4145	93.54
LSTM	0.4169	98.21	0.4006	90.40

AAE

For AAE, the best parameters for each method are shown below;

Table 4.9

Method	Standard/Adaptive	Parameter
B-Spline	Standard	$s\,val = 0.0068$
Savgol	Adaptive	$wl = 21$ & $po = 3$
DE	Adaptive	$\alpha = 0.51$ & $\beta = 0.09$
LSTM	-	seq. len. $= 5$ & acc. weight $= 0.75$

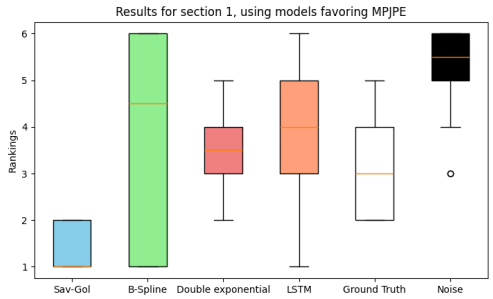
Table 4.10 Using the highest performing parameters for AAE, shown in 4.9, these results were gathered. Lower is better.

Method	MPJPE/Frame	% of Noise	AAE/Frame	% of Noise
Noise	0.4245	100	0.4432	100
B-Spline	0.4048	95.35	0.3997	90.20
Savgol	0.4175	98.34	0.3875	87.44
DE	0.4951	116.62	0.4026	90.76
LSTM	0.4304	101.38	0.3977	89.73

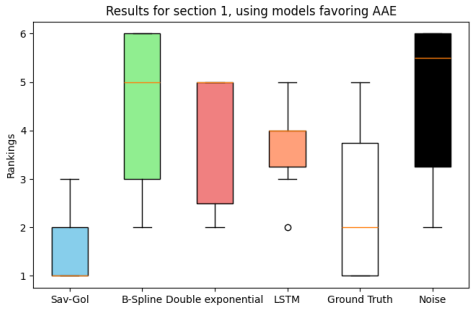
From this, we see that the best performing model is Savgol, with 12.66% of the AAE removed.

4.4 Qualitative study

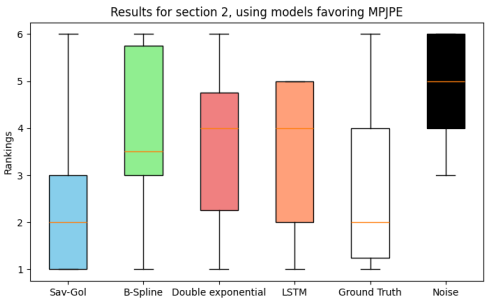
The results from our quantitative study are shown in fig. 4.4. They are presented in box diagrams marking the median rank with a red line for each smoothing model, with boxes and "whiskers" indicating the distribution of rankings.



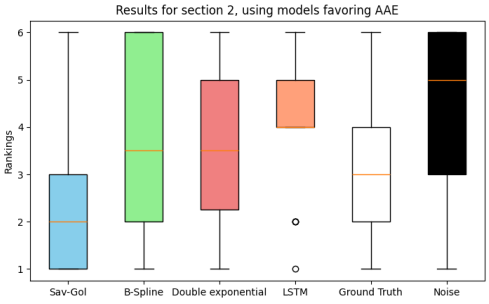
(a) Result for rankings by jitter, using models favoring MPJPE



(b) Result for rankings by jitter, using models favoring AAE



(c) Result for rankings by natural movement, using models favoring MPJPE



(d) Result for rankings by natural movement, using models favoring AAE

Figure 4.4 Results gathered in the study, visualized in box plots. Lower is better

5 Discussion

5.1 Findings

In this section the findings presented in the previous chapter will be interpreted and discussed in relation to the research questions posed in 1.2.

5.1.1 Quantitative study

From section 4.3 it can be seen that for MPJPE the B-Spline had the highest performance, reducing the largest fraction of noise in the test set. One theory for this is B-Splines ability of local control. This method has the ability to adjust how much smoothing there is in different sections based on how the control points are calculated and placed based on the data. This might give it an edge over other methods where the smoothing rate stays consistent and does not adapt to varying levels of noise in the sequence, such as Savgol.

This theory also explains our AAE results. Since the B-Spline function adapts to local patterns of the data there is a risk that the algorithm mistakenly underestimates the impact of noisier sequences and therefore not smoothing as much as necessary. In contrast, Savgol, the best performing method for reducing AAE, does not take this into account when smoothing. This results in a high degree of smoothing across the entire sequence, with the consequence of an increase in MPJPE, to achieve higher performance in AAE.

Information availability

Both double exponential smoothing and the LSTM models scored poorly using MPJPE, as seen in 4.8. One possible reason for this is that these models had access to less information regarding the surrounding points. These methods perform their smoothing using only past frames, meaning the methods are well suited for prediction purposes. This aspect is useful in real-time applications where latency is critical, as it allows for processing frames as they arrive rather than waiting for an entire sequence to process. However, having access to information about future points gives information that could be crucial to making a good estimate for denoising, which would explain why the other methods outperformed these ones in the MPJPE metric.

5.1.2 Qualitative study

With a small sample size of only 9 responses to the survey, one should be careful when interpreting the data from the qualitative study, shown in 4.4. The methods that scored best overall across all tests were the Savgol filter models. These models were also the ones to most efficiently reduce AAE in the quantitative study. This could suggest that eliminating AAE might be more important in making human motion perceived as smooth and natural, despite risking lower positional precision.

Interestingly the ground truth animation performed worse in all cases, possibly due to the jitter present in the raw motion capture data retrieved from the BEDLAM dataset in a few of the chosen animation clips. This indicates that our methods could be used for broader problems than the simulated noise described in section 3.2.

5.2 Method Analysis

In this section the design choices used during this thesis will be discussed and critiqued.

5.2.1 Dataset

The dataset used had a vast amount of animations to evaluate and train our models upon. As described in the Method chapter, each animation would be processed to add noise and the unprocessed raw animation data would be used as the baseline to compare against after having processed the noisy data. One aspect that was noted when working with this dataset was that the raw animation data was sometimes noisy to begin with. This noise would be used in training our models, which means the metrics used for evaluating results would indicate that the best method would be the one that preserves this noise. This could have unintended consequences, especially for the LSTM models, which could identify this noise as the true output, and therefore try and emulate it on input signals not containing these faults. The decision was made to ignore this issue, since only a small part of the dataset seemed to be affected.

Test set

In the test set, which was hand picked from the training set and separated from the rest of the data, this noisy raw motion capture data is heavily over-represented, being present in a third of the set. The choice to include this data in the test set was deliberate, with the intention of measuring the potential our models had at improving the noisy motion capture data. It is however possible that it skewed the results of the smoothing methods, since the test set did not match the challenges they had been trained to overcome. This is made believable when comparing the results from the LSTM in the hyperparameter optimisation validation runs, where it scored significantly better at all times compared to the test sets, even though both the external validation set and the test set had been unseen during the training of the models. This suggests that a larger test set should have been used, to more accurately represent the data of the complete set.

5.2.2 Filters

With the result in hand, there are a few points worth bringing to light about issues with the process of creating and evaluating the filtration methods. Starting off with the parameter analysis.

Initial Grid

The starting grid in table 4.1 was chosen suboptimally, based on only two animations and human intuition about which errors produced reasonable errors. Additionally, the parameters were selected assuming equal weight, w, for MPJPE and AAE, but an increase in reliability would come from using unique starting parameters for each weight. The tested parameter values and their corresponding error should have been saved and presented in the result to back up our claims of which starting grid to use.

Parameter Evaluations

There are also changes that can be made to improve the way the best parameter is calculated from the training set. As written in the discussion, currently, for each animation, only the parameter that performed the best in noted. After all animations in the training is completed, these values are summed then divided by the total number of animations. Instead, it would have been more coherent with the method comparison approach to save the error values for each parameter. This way the frame averaged MPJPE and AAE could have been used as weights to determine the best parameter, not only increasing coherency but also reducing the effect of issues that have been found to exist in a few of the animations.

Adaptive vs Standard Grid Search

If we compare the results between the standard and adaptive grid search a common trend can be seen, the parameters calculated from the standard grid search often had higher values. This is because of the functionality of our adaptive search. As previously described the parameter interval is reduced and continuously adapted following the best results over the last five animations. Since our results often show that the best parameter is closer to the lower end of our initial array of values, this adapted array will start to focus in on this area, leaving out the higher values that were tested at the start. Every now and then there comes an animation with rotational data that would benefit from having higher parameters, this is still an option for our standard method, but not the adaptive. Accumulate this rare instance over a large amount of animations and there will be a trend that the standard method produces a higher valued result. A unique behaviour can also be seen in the table 4.5. Here there is a large disparity between the standard and the adaptive parameters with weights focused on MPJPE. Our theory for this is that even though we created the adaptive method to reduce the risk of fixating on a local minimum, it instead resulted in an increase. Since we only used a few animations to determine the starting grid we hoped that the adaptive strategy could change the scope to move outside the one in table 4.1. What we instead think happened was that the first animations were similar, requiring a lower window length around a local minimum that the algorithm could not move from, resulting in worse performance.

5.2.3 LSTM

This section will discuss the design and results gathered from the LSTM models.

Rotational Acceleration Loss

One thing that behaved as expected during evaluation of the models was the effect of increasing the acceleration weight of the loss function, which in most cases lead to favoring better results for the AAE metric than for MPJPE. This effect was seen both on the external validation set and for the final models in the test set. It might seem surprising to see such a strong correlation between AAE and the loss function based on rotational acceleration given that the loss functions are based on axis-angle representations of limb rotations, while the evaluation metrics are based on 3D limb positions. However it is also reasonable to assume that jittery rotations will result in a jittery positions, even though they might be close to the correct rotation and position. This would explain why penalising the model for rotational acceleration loss would also improve the AAE of the frame.

Model size

Perhaps the most limiting factor of the model was its small size. As discussed in previous chapters, a larger size(more hidden layers and a larger hidden dimension) would probably have been able to generalize and find more complex patterns in the data that could further remove the noise and increase the quality of predictions. The small model size is also a possible reason for the model not performing better for bigger sequence lengths, which in theory would provide it with more information to make a prediction. The reason it performs worse could be that the model cannot "store" and interpret the input data for more than a couple of iterations in the rollout of the LSTM layers because of its small size. It is possible that this aspect could be improved by employing a strategy involving an autoencoder layout somewhere in the neural network architecture. This could reduce the amount of unimportant information flowing through the layers, making it easier for the LSTM layers to react and memorize longer sequences of frames.

Loss History

As seen in fig. 4.2 & 4.3, the validation loss decreases somewhat consistently with increasing epochs until plateauing. This indicates that the model cannot improve unless introduced to new training data or increasing model size, which is also why the training of each model was terminated after the validation loss had not improved for 3 epochs.

5.2.4 Qualitative study

We have a lot of afterthoughts about the survey structure. Even though a lot of planning went in to choosing animations, choosing how they would be presented, how they would be compared and rated, what should be displayed etc. it feels like if we had done it again it would look very different.

For our animations we selected two with noise in their ground truth motion capture data and two without. We later realised that even though this gave an interesting perspective on whether we could improve real world noise issues, it does not accurately reflect the dataset We also gathered some feedback from participants regarding their thoughts about the survey. There was a common trend in the survey being straining and difficult to finish for a couple of reasons. Many animations looked similar, making them hard to rank. the jitter interfered with the section on natural movements causing a bias towards those who remove jitter effectively. Finally, the white characters in an endless void (see fig. 3.2) made all movements feel unnatural and robotic.

Another point worth discussing further is that the animation representing ground truth performed worse than expected, especially on the section regarding natural movements without ground truth noise. Our theory is that since four of the six animations all displayed the same pose errors, the participants were skewed to think

this error was natural due to the majority of characters performing the same (erroneous) movements. One animation specifically had a large pose error included in the animation. Even though this error makes a person move their leg through the floor this did not feel unnatural according to the surveyed, since the floor was invisible.

5.2.5 Bevy

The usage of BEVY resulted in an increased amount of work. Although this engine is fun to work with and has a high potential, using BEVY whilst being new to RUST proved to be a challenge in many occasions. Thankfully, our supervisor Erik was very knowledgeable and could help us when we found no solution available. It would have been wise to use another engine for this thesis simply because of the time constrains, but the functions we needed were available so the project could be completed. The most common issue though, was that information about solutions to relevant issues had outdated answers. This is a side effect of the engine still being new and under development, so this downside will eventually not be as relevant but resulted in some hindrance for us. We mainly used BEVY for visualization purposes, but it was also used to gather the 3D positional data from rotations. This caused confusion when results for the quantitative study were gathered, since with the same rotational inputs, the output in 3D coordinate data had slight variations depending on which of our computer ran the simulation.

5.2.6 Future Work

Filters

When writing, we have seen potential future developments that would be of interest to study and investigate. Currently, we focus just on removing noise from the rotation vector inputs. An interesting aspect would be if there are any improvements that can be made by filtering the positional coordinates instead. As mentioned previously in the discussion, an improvement to both the initial parameters and the adaptive grid search algorithm can be considered. When done right out theory is that this should not only increase reliability that the chosen interval contain the global maximum, but the adaptive search should consistently perform better than just the standard.

Neural Network

As explained before, one bottleneck for the LSTM models was the model size, which was capped due to long training times. This would be a smaller issue with more computational power, which could decrease the time required dramatically. Exploring solutions involving cloud computing could be beneficial in this aspect, perhaps allowing for bigger models and larger datasets. Another path to explore would be performing more data preparation, for example normalisation along the input sequences or using dropout during training to hide certain connections between layers in the network to improve the generalisation of the model.

General Areas of Interest

The LSTM predictions generally did a better job filtering out pose errors created by our noise model, but worse in filtering out the high frequency noise like jitter. The classic filters on the other hand, were much better at reducing these errors whilst ignoring the pose errors. It would be of interest to combine the methods used in the thesis, to see if there is a way to create a combined solution that the effective at solving both types of issues.

The noise modeling is another field where more work can be done to more accurately represent the problems present in 3D pose estimation. The issues emulated in the current model are present in actual pose estimation data, but one aspect that was largely ignored is lost frames where the estimator loses track of the subject being tracked. This problem is particularly common when more subjects are present in the same scene, and partially occlude one another. This type of data could be emulated by removing frames at certain times. The models would then have to work with these new type of errors, and try to predict the actual pose for the missing frames based on the remainder of the sequence.

6 Conclusion

One of the questions we set out to answer in this thesis was whether it is possible to remove noise from 3D human pose estimators in such a way that it improves the user experience. Looking at our results, we find that for both our quantitative metrics, we were able to reduce the errors created by unwanted noise with 7.15% and 12.66% for MPJPE and AAE respectively. The qualitative study gave a clear indication that all methods we tested consistently ranked higher than the noisy animation on both the amount of jitter and the natural movement criteria. One of the methods, Savgol, outperformed even ground truth in many scenarios, although this comparison to ground truth should be interpreted with caution because of the reasons mentioned in the discussion. Compared to our simulations of how real pose estimation data would look, our conclusion is that our methods improve the user experience.

Another aim was to investigate and compare an LSTM network with the classic filters to see if there is any benefit to selecting either of these methods. The results in all of our studies point to the LSTM models performing worse in solving the proposed problems. The reasons for this could be that the LSTM is solving a slightly different problem than the top performing filters, namely prediction based on past input, whereas the filters use information from future *and* past points in time to smooth out the input signal. Another explanation could be that the LSTM models were poorly designed or simply too small to capture the patterns present in the dataset.

Bibliography

Black, M. J., P. Patel, J. Tesch, and J. Yang (2023). "BEDLAM: a synthetic dataset of bodies exhibiting detailed lifelike animated motion". In: *Proceedings IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 8726–8737.

Bridgeman, L. (2019). "Full-body performance capture of sports from multi-view video". In: URL: https://api.semanticscholar.org/CorpusID:210990220.

Chevalier, G. (2021). *Image from wikipedia*. CC BY-SA 4.0. Licensed under Creative Commons Attribution-ShareAlike 4.0 International License. URL: https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg.

Chung, M. G. and S.-K. Kim (2013). "Efficient jitter compensation using double exponential smoothing". *Information Sciences* **227**, pp. 83–89.

Curtright, T. L., D. B. Fairlie, and C. K. Zachos (2014). "A Compact Formula for Rotations as Spin Matrix Polynomials". *SIGMA* **10**, p. 084. DOI: 10.3842/SIGMA.2014.084. arXiv: 1402.3541 [math-ph].

Dam, E. B., M. Koch, and M. Lillholm (1998). *Quaternions, interpolation and animation*. Vol. 2. Citeseer.

De Boor, C. (1976). *Splines as linear combinations of B-splines: A survey*. University of Wisconsin-Madison. Mathematics Research Center, p. 9.

Feurer, M. and F. Hutter (2019). "Hyperparameter optimization". In: pp. 3–33. ISBN: 978-3-030-05317-8. DOI: 10.1007/978-3-030-05318-5_1.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Graßhof, S., M. Bastholm, and S. Brandt (2023). "Neural network-based human motion predictor and smoother". *SN Computer Science* **4**. DOI: 10.1007/s42979-023-02195-0.

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory". *Neural Computation* **9**:8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Holding, D. (2024). "Unrolling rotations". https://theorangeduck.com/page/unrolling-rotations.

Ibrahim, M. F., M. Y. Misro, A. Ramli, and J. M. Ali (2017). "Maximum safe speed estimation using planar quintic bézier curve with c2 continuity". In: *AIP Conference Proceedings*. Vol. 1870. 1. AIP Publishing.

Lipton, Z. C., J. Berkowitz, and C. Elkan (2015). *A critical review of recurrent neural networks for sequence learning*. arXiv: 1506.00019 [cs.LG].

Liu, Y., B. Dang, Y. Li, H. Lin, and H. Ma (2016). "Applications of Savitzky-Golay filter for seismic random noise reduction". *Acta Geophysica* **64**:1, pp. 101–124.

Loper, M., N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black (2015). "SMPL: a skinned multi-person linear model". *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* **34**:6, 248:1–248:16.

Memar Ardestani, M. and H. Yan (2022). "Noise reduction in human motion-captured signals for computer animation based on b-spline filtering". *Sensors* **22**:12. ISSN: 1424-8220. DOI: 10.3390/s22124629. URL: https://www.mdpi.com/1424-8220/22/12/4629.

Mishra, P., A. Karami, A. Nordon, D. N. Rutledge, and J.-M. Roger (2019). "Automatic de-noising of close-range hyperspectral images with a wavelength-specific shearlet-based image noise reduction method". *Sensors and Actuators B: Chemical* **281**, pp. 1034–1044. ISSN: 0925-4005. DOI: https://doi.org/10.1016/j.snb.2018.11.034. URL: https://www.sciencedirect.com/science/article/pii/S0925400518319889.

Muraina, I. (2022). "Ideal dataset splitting ratios in machine learning algorithms: general concerns for data scientists and data analysts". In: *7th International Mardin Artuklu Scientific Research Conference*, pp. 496–504.

Shao, Q., J. Wang, B. Zhou, V. A. Tran, G. Krishnan, and S. Nayar (2023). "Neuro predictor: a neural network approach for smoothing and predicting motion trajectory". *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* **7**:3, pp. 1–25.

Song, H., S. Heo, J. Kang, and S. Lee (2015). "3d character animation: a brief review". *Journal of International Society for Simulation Surgery* **2**:2, pp. 52–57.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". *Nature Methods* **17**, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Wang, J., S. Tan, X. Zhen, S. Xu, F. Zheng, Z. He, and L. Shao (2021). "Deep 3d human pose estimation: a review". *Computer Vision and Image Understanding* **210**, p. 103225. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2021.103225. URL: https://www.sciencedirect.com/science/article/pii/S1077314221000692.

Master's Theses in Mathematical Sciences 2024:E48
ISSN 1404-6342

LUTFMA-3547-2024

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/