Long short-term memory networks for body movement estimation

Sander Vlutters, s.vlutters@student.utwente.nl

***Abstract*—This paper introduces an approach to reconstruct full body motion from a small set of inertial sensors using a long short-term memory (LSTM) network. Although a small set of sensors provides incomplete information for this task, missing degrees of freedom are estimated based on pre-recorded full body motions. Several hyperparameters were tested and their effects on the LSTM's capabilities of synthesizing full body motion were evaluated and compared with a standard feedforward neural network (FFNN). Our results show that the LSTM performs no better than a FFNN on this problem, indicating that information from the past is unhelpful for estimating missing degrees of freedom. Also, it was found that the networks have difficulties making correct estimations when excluding positional information from our set of sensors as estimator.**

***Index Terms*—Recurrent neural networks, long short-term memory, deep learning, body movement estimation, motion capture.**

October 17, 2016

I. Introduction

THE process of capturing and reconstructing full-body movement with high-quality motion capture (mocap) technologies often requires many sensors or markers to be strapped to each body segment. As a consequence, these mocap technologies are often far too expensive for private usage. Also, they are inconvenient in use as it takes much time to suit up. These problems could possibly be relieved if the number of sensors can be reduced to a small number as long as the ability to reconstruct full-body movement remains without suffering from major reconstruction errors. Although the usage of only a small number of sensors provides insufficient information to capture full-body poses, missing information could be estimated based on pre-recorded full-body mocap data. This is based on the belief that natural human motion is highly coordinated and the existence of dependencies between (the positions of) limbs during movement [1] [2] [3]. Modeling these dependencies is difficult however due to the high complexity and dimensionality of human movement. Previous works that have solved this problem include that of Chai and Hodgins [1] and Liu et al. [4][5] who successfully adopted the k-nearest neighbor method. Although these authors have managed to achieve good results, their approach also faces several limitations. First, their approach assumes the knowledge of positional data in order to make predictions. While capturing positional data is no problem for e.g. marker-based mocap systems, accurately capturing this information is much harder for inertial mocap systems due to the occurrence of positional drifts. Since the goal of this work it to obtain an inertial mocap system using only a few sensors, it will be looked into whether positional data can be omitted for sake of other (lower level) features as a basis for predicting missing information. Other limitation of the k-nearest neighbor method include the fact that it requires an online database containing the expected motions at all times and the problem of scaling badly when the size of the database increases. To avoid these limitations, we turn to the area of deep learning, which has demonstrated its effectiveness in recent years by beating the state-of-the-art in numerous pattern recognition tasks [6]. The deep learning architecture most suitable for our problem is the recurrent neural network and with that in particular the LSTM variant. This architecture was chosen for its capability of handling sequences of inputs, which was believed to be invaluable since natural human movement is 'smooth'; we known that features such as orientations, positions, accelerations, velocities, etc are limited to a maximum rate of change over time, making the targeted output likely more predictable given knowledge of the past. Using the LSTM network, several different hyper-parameter were tested such as how far the LSTM looks back in time and the kind of features that were used as input. Given the results of these tests, we took the network with the best performing settings and compared it with a standard FFNN using the same hyper-parameters (for as far as possible) to determine whether the problem benefits from a recurrency. Next to that, we compared the performances of networks using positional data as input with networks excluding this data to determine whether this feature can be omitted. By making these comparisons, the following research questions will be answered:

1) Does the problem benefit from a recurrency?
2) Can positional data be omitted from the input, possibly for sake of other features, without suffering in performance?

Our results show that for this problem, the performance of the LSTM is on-par with that of a FFNN, regardless of the depth with which the LSTM propagates back through time. These findings indicate that the problem has little to no benefit from a recurrency with our settings. Furthermore, we have found the absence of positional data to have a negative effect on each network's performance.

The rest of the paper is structured as follows: section II provides background information about mocap and its problems, the recurrent neural network and the LSTM. This is followed by section III in which we provide an overview of works that have dealt with the problem of full-body motion estimation using few sensors and works that have applied RNNs on mocap problems. After that, we introduce our approach in section IV which includes a description of the dataset, how features are

normalized, the architectures and hyper-parameters that were tested, how we synthesize motion from a network's output as well as how performances are measured. Section V then presents the results, where we compare the performance of a LSTM with that of a standard FFNN and where we compare networks using positional data as input with networks omitting this data. Finally, the paper is wrapped up with a discussion in section VI, conclusions in section VII and suggestions for future work in section VIII.

II. Background

A. Motion capture

Motion capture has been an active field of research for decades and has found its use in a wide range of domains, including entertainment, medical applications, robotics, sports, virtual reality and more. To this day however, there are a number of drawbacks with mocap pending a solution. These drawbacks are not only restricted to the aforementioned such as high cost and being cumbersome in use, but also include problems like being restricted to the space it is operated in (as a consequence of sensitivity of equipment to electro-magnetic interference or a camera requiring a specific field of view), the possibility of occlusions, markers/sensors moving during capturing, degrees of freedom offered by the system among others. These drawbacks may not only prevent mocap systems from possibly being suitable for commercial use, they might also hinder it from being used in other potential applications. In order to overcome some of the aforementioned limitations, a vast array of different mocap systems have been developed over the years, ranging from optical systems to mechanical, acoustic, magnetic, inertial and more [7]. Unfortunately, many of these approaches are not suited for private use. One example of a system that has successfully introduced mocap for private use is Microsoft's Kinect [8], which uses a single, low-cost, depth camera which is capable of tracking full-body motion. Our proposed method in contrast focuses on capturing full-body motion using an inertial system such as that found in Roetenberg et al. [9], but reduced to contain only a small number of sensors. Unlike visual mocap systems, these systems are less restricted to the space in which they are used and they do not suffer from occlusion problems.

B. The recurrent neural network

The recurrent neural network (RNN) is an adaption to the standard FFNN to allow it to process temporal information by allowing information from previous inputs to persist in the network. This is achieved by adding feedback loops to neurons, granting them the ability to pass the results of previous events to future inputs and to utilize this in making decisions. As a result, the rules for calculating the network's output are also slightly different from that of a FFNN. First, instead of taking a single vector x as input, we take a sequence of vectors $x_1, x_2, .., x_T$ as input. Given this sequence, we can calculate the activations in the hidden layer at each timestep t as follows:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{1}$$

In this equation, W_{hh} denotes the weights of the feedback loops going from the hidden layer to itself, W_{hx} denotes the weights going from the input layer to the hidden layer, b_h is the bias while σ denotes an elementwise application of the normalization function (such as the sigmoid). Also note how the equation is recursive since we add the activations of the last timestep, h_{t-1}, to the current input x_t. Therefore, by applying this equation recursively the activations at each timestep can be calculated. Using the calculated activations, the output at timestep t can then be computed by multiplying the activations h_t in the hidden layer with the weights W_{yh} going from the hidden layer to the output layer, according to the rule:

$$y_t = W_{yh}h_t \tag{2}$$

Although the process of calculating the output is performed recursively, training a RNN is similar to training a FFNN and is per usual done with a variant on the backpropagation algorithm [10]. In this variant, called backpropagation through time (BPTT), gradients are backpropagated through the feedback loops by unfolding the network through time. To see how this works in practice, see Figure 1 which shows a chunk (e.g. a neuron) of a network at a timestep $t = T$ being unfolded for T timesteps back in time. Like standard backpropagation, BPTT consists of a repeated application of the chain rule, with the main difference being that gradients are not only dependent on the output layer, but also on the influence of the hidden layer at each timestep. Therefore to update the weights in the hidden layer, we start by calculating the error at timestep $t = T$ using some cost function (such as the mean squared error), and calculate each partial derivative recursively using the obtained error. Since the weights in the feedback loops are the same at every timestep, we can sum the partial derivatives over all timesteps per hidden unit. By doing so, we obtain their gradients which can subsequently be used to update the weights. While this training method may look fine on the surface, training a standard RNN can be very difficult. One of the reasons for this difficulty is the fact that errors are multiplied with each other in each timestep due to the recurrency in Equation 1. As a result, when errors are small, they quickly diminish. On the other hand, when errors are large, they can quickly grow very large. Consequently, weights are updated disproportionately and the network fails to capture long-term dependencies.

C. The long short-term memory network

Several variants on the RNN have been proposed in order to overcome its shortcomings. From these variants, the most popular are the long-short term memory network (LSTM), first introduced by Hochreiter and Schmidhuber [11] and the more recent gated recurrent unit (GRU), introduced by Cho et al. [12]. In this section, the LSTM will be discussed, which will be used throughout this paper as it has established itself as the most popular variant on the RNN while it has also been proven that the GRU does not outperform the LSTM [13]. The LSTM is much like the standard RNN, with the most notable difference being that they replace ordinary hidden neurons with special units called memory cells. See Figure 2 for an

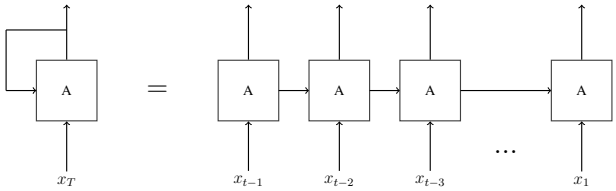


Fig. 1: Unfolding a chunk A from a RNN at timestep $t = T$ for T timesteps back in time.

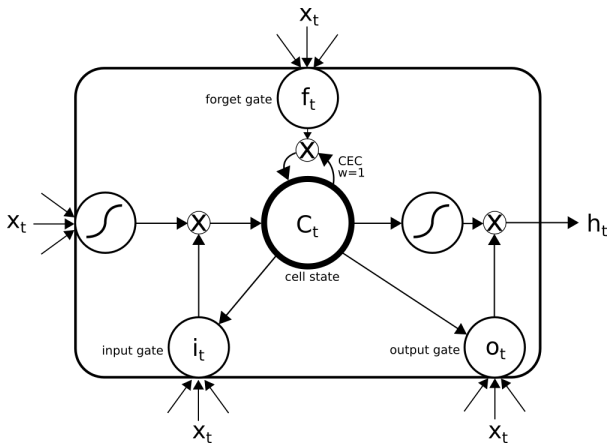


Fig. 2: The composition of a single memory cell. Here the nodes with the $\times$-sign denote activation functions, whereas nodes with the 'S' shape denote normalization functions.

example of a memory cell. Each memory cell is a composition of multiple units. The first and most import unit is the cell state denoted by C_t in the figure. This is a neuron which stores a value over time. It has a self-recurrent connection that allows the cell state to either remember or forget a value from a given timestep. This self recurrent connection is often called the Constant Error Carousal (CEC) and has a fixed weight of 1. The CEC solves the vanishing/exploding gradient problem by enforcing a constant error flow from one timestep to another. By doing so, LSTM's are capable of learning the importance of events that happened thousands of time steps ago [6]. Since errors can easily flow through the CEC, additional units are added to the memory cell to regulate which information is allowed to flow into or out of the cell state. For this purpose, 3 gates are added, namely an input, output and forget gate, each one interacting with the cell state in its own way. The first step in the memory cell is to decide which information from previous timesteps is allowed to persist in the cell state and which information should be thrown away. This is regulated by the forget gate. Next to that, the memory cell needs to decide on the importance of new information from the current timestep. This is regulated by the input gate. By combining the flows from these two gates, we obtain the following update rule for the cell state c_t at timestep t:

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c) \quad (3)$$

Here f_t and i_t are the outputs of respectively the forget and the input gates. All gates output a value between 0 and 1. Note that when for example f_t outputs a 0, it means that all information from previous timesteps is dropped, whereas if it outputs a 1, all information from previous timesteps is retained. Similarly, i_t determines the importance of new information. This new information is specified by the term $W_{xc} x_t + W_{hc} h_{t-1} + b_c$, which is similar to Equation 1. In this term, we simply add the activation of the input at the current timestep with the activation from the previous timestep. The normalization function for this term is chosen to be the hyperbolic tangent (tanh) function, which puts it in the range $[-1, 1]$. This has the benefit that values in the cell state can either be increased (if important) as well as decreased (if unimportant) when combining them with the information obtained from the previous timesteps (that is $f_t c_{t-1}$). Now that the activation function of the cell state has been determined, we specify the functions for the gates. These are specified by the following equations:

$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f) \quad (4)$$

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i) \quad (5)$$

$$o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + W_{co} c_{t-1} + b_o) \quad (6)$$

Here f, i and o are respectively the forget gate, input gate, and output gate. Again, these equations are somewhat similar to Equation 1, with the difference being that the output of the last cell state is now also included. Here we take the current input x_t, the activation of the previous timestep h_{t-1} and the activation of the last cell state c_{t-1}, and combine them after multiplying them with their respective weights. Next to that, the σ denotes the sigmoid function in these 3 equations such that the gates produce outputs in the range $[0, 1]$. Finally, the actual output of the memory cell needs to be decided. For this purpose, the output gate is added. This gate essentially filters the output of the cell state after it has been normalized according to the following equation:

$$h_t = o_t tanh(c_t) \quad (7)$$

III. Related work

A. Motion estimation from a reduced sensor set

While the problem of estimating full human body motion on the basis of a small number of sensors is found increasingly more often in the literature, only few related works are available to this date. This section therefore serves the purpose of giving an overview of works dealing with the same or resembling problems. Most works dealing with this

problem have adopted the k-nearest neighbor search, often in combination with PCA to reduce the dimension of human motion. Works following this approach include that of Chai and Hodgins [1], who uses a setup consisting of a subset of 6-9 retro-reflective markers in combination with two synchronized video cameras to estimate full-body poses using positional data. This is performed using a heterogeneous database consisting of 10 different full-body exercises measured with a set of 41 markers. Exercises in their database include walking, running, hopping, jumping, boxing and Japanese sword art. For estimating full-body motion with their reduced marker set, they perform queries on their database for poses most fitting for the current motion. More recent work following the same approach include that of Liu et al. [5], who uses a set of 6 inertial sensors placed at the ankles, hands, torso and head to estimate full-body motions using positional and orientational data. Exercises in their database include boxing, golf swinging and table tennis. Using this approach, Chai and Hodgins as well as Liu et al. report positive results, capable of synthesizing a naturally looking animation comparable in quality to those obtained from a commercial mocap system. Similarly, the work of Tautges et al. [14] also uses the k-nearest neighbor method, but instead uses acceleration data obtained from a set of 4 accelerometers placed at the wrists and ankles for predictions. Although these works report positive results, their approach of using a k-nearest neighbor search also faces problems such as requiring an online database containing the expected behaviors at all times as well as scaling badly when the size of the database increases. By using an artificial neural network instead, we're not bound to a database once the network has been trained. Furthermore, we expect neural networks to be more robust at estimating poses that are not necessarily contained in the dataset. This belief is based on the fact that the k-nearest neighbor method lacks any form of parameterized function, making it less capable of generating new outputs as the output is simply obtained by averaging over the k nearest samples in the dataset. A neural network on the other hand represents a parametric function, capable of generating outputs that may not be able to be obtained by averaging over samples in the dataset.

B. Motion estimation neural networks

To the best of our knowledge, no previous attempts have been made that use (deep) artificial neural networks to estimate full human body motion using a small number of sensors. There are however numerous papers to be found that use (recurrent) neural networks for forecasting or recognizing human action. Fragkiadaki et al. [15] uses a LSTM with 3 hidden layers in addition to a restricted boltzmann machine to forecast motion for up to 600 miliseconds. Forecasting poses is tested in two different settings; One using a dataset of mocap data and one using data collected from video. Activities that the network is trained on include walking, eating and smoking. Next to forecasting motion, numerous papers can be found that apply RNNs for classifying motion. Unfortunately, most of these papers focus on motion captured by video rather than from a mocap system, making them only very loosely related. For a paper that uses mocap data instead, we refer to Bitzer and Kiebel [16] who uses a standard RNN in combination with PCA to successfully classify three different walking styles (childish, depressed and shy).

IV. Method

A. Dataset

The dataset used contains the full-body movements of 6 participants (3 men and 3 women) performing 16 different activities. The performed activities include physical exercises and other activities of daily living such as walking with a cup of water in hand, running, squats, etc. Each activity was executed 3 times in succession, totaling up to a dataset with $6 \cdot 16 \cdot 3 = 288$ trials in total. Measurements were done using Xsens' MVN Link [9], a mocap system containing a set of 17 Inertial Measurement Units (IMUs) which maps their outputs to a human model containing 23 segments. Each IMU is composed of a 3D accelerometer, a 3D magnetometer and a 3D gyroscope, offering multimodal sensor information at a sampling rate of 240Hz. Captured features include the orientation, position, (angular) velocity and (angular) acceleration for each IMU. Despite the fact that these kind of sensors provide a lot of different information, the only features of interest are the orientation, angular velocity and angular acceleration. The positions, velocities and accelerations are omitted as they are not reliable enough due to drift and the fact that they are dependent on the length of the person. Now that features have been determined, we also select a subset of 5 segments from the 23 segments of the human model which will form the basis for estimating the remaining 18 segments. For the remainder of this paper, we will call these subsets for in- and output respectively S_x and S_y. The segments chosen for S_x correspond with the forearms, the lower legs and the pelvis. See also Figure 3 depicting the positions of the segments in these subsets. The segments in S_x were chosen under the assumption that they hold the most information about the segments in S_y. Reasoning behind this choice is that the features of the forearms and lower legs are dependent on the features of respectively the upper arms and upper legs, whereas the reverse is true to a far lesser extend. Furthermore, we suspect that the segments located at the hands, feet and toes are too independent from the rest of the body to form reliable predictors. Next to that, similar to Chai and Hodgins [1], we chose the pelvis as centralization point for normalizing the features of the other segments as the pelvis is the most stable segment during movement, holding the most reliant information about the general orientation of the body. This way, the features relative to the pelvis can be calculated such that global translations and rotations of the body are removed. Finally, now that useful features and a set of predictors have been determined, a test- and training dataset need to be determined. As we are interested in the feasibility of estimating motion regardless of user or the kind of exercise that is performed, we split the dataset by person and train the neural networks on all exercises from the group of training persons. Although it is suspected that higher performances may be attained by training networks per

person and/or exercise as it lowers the amount of variance between samples, that is not the goal. Therefore the networks are trained on all trials performed by all participants in the training dataset to obtain a general system. Here the training set is obtained by treating participants 1-5 as the train subjects while treating person 6 as the test subject. This way, it can be tested how such a system would perform on a new person, similar to how it would be in a real-life setting.

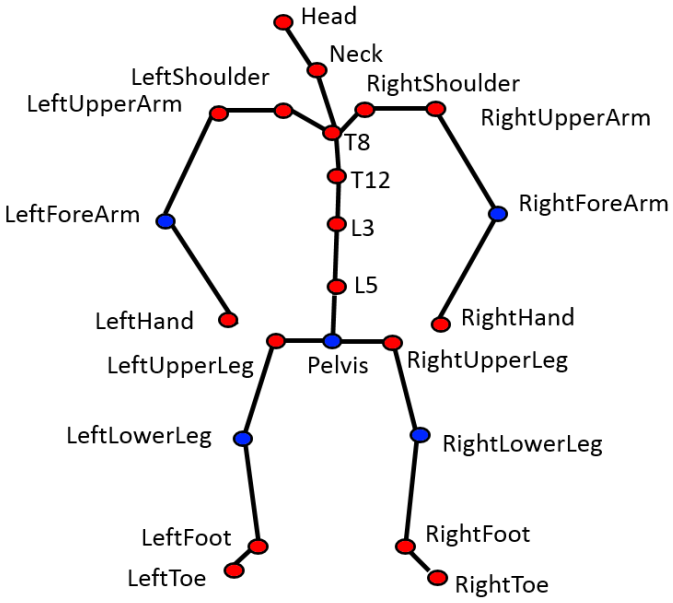


Fig. 3: Locations of the segments as captured by the mocap system. The segments denoted by blue form the subset S_x and are used for estimation. The segments denoted by red form the subset S_y that is to be estimated.

B. Feature normalization

Before the selected features can be fed to a neural network, some pre-processing and standardization needs to be applied. Currently, there are several problems with using the raw features. First, all features are based on a global translation and rotations of the body, meaning that the values of the features change depending on the general direction of the measured person. As a consequence, when for example a person rotates around his/her Z-axis (such that the general direction that person is facing changes), then the features' values will also change accordingly. This has the undesired effect that a person's general direction is included in the features whereas this is independent of his/her body posture. In order to translate the features such that they lose this dependency, from the set of segments, one segment is chosen as a centralization point. For this purpose, the segment located at the pelvis was chosen as it is located near the center of the body which is relatively stable during movement. Using the orientations of the pelvis, the features of the remaining segments can be translated such that they are relative to the pelvis, removing global translations and rotations in the process. Following these translations, each of the features need to be normalized to make them suitable for a neural network. How this normalization is performed is dependent on the kind of feature. Followed is a description per feature of its normalization process. These normalizations are thus applied on the features relative to the pelvis:

1) Orientations: The orientations are represented by quaternions, which have two representations for any given orientation, namely $Q = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}$ and $-Q = \begin{bmatrix} -q_1 & -q_2 & -q_3 & -q_4 \end{bmatrix}$, such that $Q = -Q$. As a consequence, each orientation may be represented by one of the two representations, which is problematic as it hinders the neural network from learning a mapping from in- to output. In order to deal with this problem, each quaternion is converted to the same representation. This is done by checking the term q_1 for negativity and negating the whole quaternion if this holds true. As the values of unit quaternions are already in the interval of $[-1, 1]$ (with the exception of q_1, which is now in the range of $[0, 1]$), no further normalization is required.

2) Angular velocity and angular acceleration: As it is desired that a change in one feature has the same amount of influence on the output as an equal change in another feature, each feature must be translated to the same range of $[-1, 1]$. In order to rescale features into this range, their value range must be known. However, the problem with features such as the angular velocity and angular acceleration is that their value range is practically unbounded. For this reason, we measured the value distribution of the features relative to the pelvis over the entire dataset and determined their boundaries accordingly. See Figure 4 for histograms containing the frequency distribution of the features relative to the pelvis for the segment corresponding with the left forearm over the entire dataset. The frequency distributions of other segments in S_x revealed to have a similar distributions, but were left out for organizational reasons. Based on these figures, the value ranges for the angular velocity and angular acceleration were determined to be respectively $[-10, 10]$ (m/s) and $[-50, 50]$ (m/s^2). Using these boundaries, each value is rescaled to $[-1, 1]$ according to the following equation:

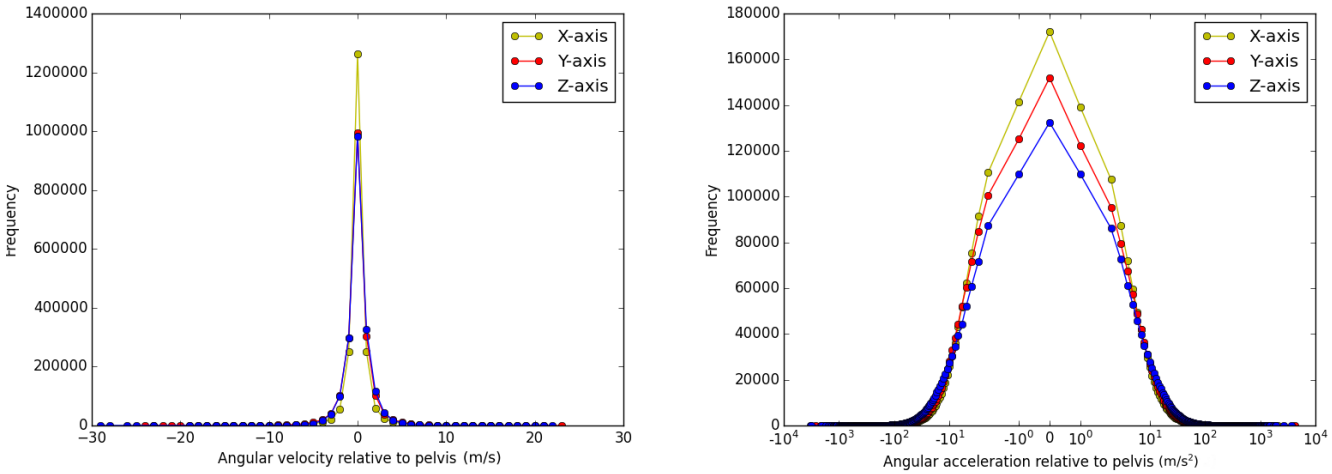
$$f(e, a, b, c, d) = \begin{cases} c, & \text{if } e \leq a \\ d, & \text{if } e \geq b \\ \frac{e-a}{b-a} \cdot (d - c) + c, & \text{otherwise} \end{cases} \quad (8)$$

This equation rescales an element $e \in \mathbb{R}$ by applying the mapping $[a, b] \mapsto [c, d]$. Here elements e are expected to fall in the range of $[a, b]$. Elements that happen to fall outside this range are cut off to the nearest value in this range.

C. LSTM architectures

This section describes the LSTM networks that have been tested and their hyper-parameters. Several parameters were varied and their impact on the network's performance was measured to determine their optimal settings. Tested parameters include:

1) The number of timesteps that the LSTM will propagate back through. For this parameter, we have tested the values $\{1, 5, 10, 15, 20, 25, 30\}$, which correspond with backpropagating through $\{0, \frac{1}{6}, \frac{2}{6}, .., 1\}$ seconds.



(a) Angular velocity left forearm

(b) Angular acceleration left forearm

Fig. 4: Value distributions of the angular velocity and angular acceleration relative to the pelvis over the whole dataset. Each value is grouped under the integer closest to it, i.e. values in the range $[-0.5, 0.5)$ are grouped as 0, values in the range $[0.5, 1.5)$ are grouped as 1, etc. The above distributions are based on the segment corresponding with the left forearm. The other input segments show similar distributions.

Although our mocap data is sampled at 240Hz, we downsample it to 30Hz by taking only every 8th frame. Reason for doing this is twofold. First, consecutive frames will be very similar to each other at 240Hz, therefore likely adding little benefit to a LSTM when backpropagating through time. Second, by decreasing the sampling rate, it becomes easier and computationally less heavy for a LSTM to look further back in time. Once the trials have been downsampled, a zero-padding needs to be applied before each sequence of frames obtained from a trial. This is performed to allow the LSTM to backpropagate through t timesteps at all times, which is needed considering the first frame of each trial does not have any predecessors. In this process we add $t-1$ vectors containing only zeros in front of each sequence of input vectors $\left[v_1, v_2, .., v_n\right]$ obtained from a trial to obtain a new sequence $\left[v^z_{-t+2}, v^z_{-t+3}, ..v_1, v_2, .., v_n\right]$ where a vector v^z_i denotes a zero vector. Samples for the LSTM are then obtained using a sliding window of length t starting at v^z_{-t+2} to v_1 and shifting one position at a time. Note how each sample is consequently a sequence of t vectors inherently containing all information from the past.

2) The size of the hidden layer. Two different settings for the number of hidden units were tested, namely a smaller network containing 30 hidden units and a bigger network containing 100 hidden units.
3) The features used as input. Two different inputs were tested, namely:
   a) (relative) Orientations of the segments in S_x
   b) (relative) Orientations, angular velocity and angular acceleration of the segments in S_x

   These features were chosen under the assumption that orientations are essential as input whereas it was uncertain whether features that explicitly indicate a rate of change with respect to time such as the angular velocity and angular acceleration will improve performance. These features were therefore tested separately from each other.

Given the inputs, we train all networks to output the (relative) orientations of the segments in S_y. Reason for using only the orientations as output is twofold. First, the orientations (i.e. quaternions) already fall conveniently in the range of $[-1, 1]$, making them suitable as targeted output. Second, forward kinematics can be applied to translate the segments' orientations to positions such that full-body poses can be derived. Using these in- and outputs, the samples are divided into mini-batches to speed up training and reduce overfitting. For the mini-batches a size of 200 is chosen, following the guidelines proposed by Bengio [17] who suggests to choose this value between 1 and a few hundreds. Furthermore, each network is trained for a maximum of 100 epochs, which was found to be sufficient. In each epoch, the order of the samples in the trainingset is randomly shuffled. Recall that a single sample consists of multiple vectors accumulated over t timesteps, therefore retaining temporal information when shuffling. Shuffling is done to avoid the networks from adapting too much and possibly favoring exercises seen in the most recent trials over exercises seen in previous trials. Also, we try to avoid any other overfitting by adding a dropout to the connections from the input layer to the hidden layer. For this parameter, a value of 0.5 is chosen as it is close to the optimum for a wide range of networks and tasks [18]. Additionally, in these same connections we use the rectified linear unit (ReLu), described by $f(x) = max(0, x)$, as activation function as it is fast and as it does not suffer from the vanishing gradient problem [17][19]. Next, going from the hidden layer to the output layer, a linear activation function is used since the networks

are trained to perform a regression. For the same reason, we use the mean squared error cost function. Finally, all networks are trained using the rmsprop optimization function [20] as benchmarks have shown that it works very well for training RNNs [21].

D. Post-processing

With the neural networks trained to estimate relative orientations, some translations need to be applied in order to derive full-body postures from the network's output. First, given that the acquired output may contain estimation errors it is likely that the estimated orientations (quaternions) are not of unit length as conform the rules of a quaternion. Therefore, each quaternion $Q = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}$ is first normalized such that they have a magnitude of 1, where the magnitude is defined as $m = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}$. Once normalized, each orientation is translated back according to the pelvis to obtain an estimation of the original, global, orientation. Now that the network's output is translated to usable orientations, the results are merged with the orientations from the input set S_x to obtain the orientations of all segments. Finally, given all orientations, the knowledge of the segment lengths can be utilized to obtain their positions and to construct a full-body posture.

E. Distance measure

In order to determine a network's performance, a distance measure is needed since a network's estimation error does not necessarily reflect its ability to synthesize accurate motion. Reason for this twofold. First, a low average estimation error does not automatically indicate that the estimated orientations are in fact close to the real orientations. Consider for example an estimated orientation (quaternion) defined by its 4 terms q_1 to q_4. Even when 3 of the 4 terms are estimated correctly one of the terms may be completely off from the real value, yet resulting in a relatively low average error. This may ultimately have a much bigger impact on the estimated body pose than in the case when all 4 terms are equally off with the same average error. Second, when translating orientations to positions, the orientations of some segments have a much bigger impact on the resulting body pose than others. For example, an estimation error in the upper leg affects the positions of the segments in the whole leg, whereas an estimation error in the foot has a much lesser impact. For these reasons, instead of using the network's estimation errors as performance measurement, we measure the average Euclidean distance between the positions synthesized by the network's output and the original positions over the whole test set. In mathematical terms this is formulated as follows. Assume our test set consists of n trials $T_1, T_2, .., T_n$ and assume that a given trial T_i (with $1 \leq i \leq n$) contains m_i frames $f_1, f_2, .., f_{m_i}$. Now each of these frames contains the positions of 23 segments, $s_1, s_2, .., s_{23}$. Let's denote the estimated position of segment s_k (with $1 \leq k \leq 23$) at frame j (with $1 \leq j \leq m_i$) of trial i as $\begin{bmatrix} x'_{ijk}, y'_{ijk}, z'_{ijk} \end{bmatrix}$ and the real position at this same frame as $\begin{bmatrix} x_{ijk}, y_{ijk}, z_{ijk} \end{bmatrix}$. Then we calculate the average Euclidean distance over the test set as follows:

$$\frac{1}{23 \cdot (\sum_{i=1}^{n} m_i)} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{k=1}^{23} \sqrt{(x'_{ijk} - x_{ijk})^2 + (y'_{ijk} - y_{ijk})^2 + (z'_{ijk} - z_{ijk})^2}$$

Here in the term $\sum_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{k=1}^{23}$ we first iterate over all n trials, followed by iterating over all m_i frames contained in the current trial i and finally we iterate over all 23 segments that are contained in each frame. Next, we calculate for each segment the Euclidean distance between its estimated and real positions in the term $\sqrt{(x'_{ijk} - x_{ijk})^2 + (y'_{ijk} - y_{ijk})^2 + (z'_{ijk} - z_{ijk})^2}$. Finally, once the Euclidean distances of all segments contained in the dataset have been summed up, we calculate the average by multiplying the result with $\frac{1}{23 \cdot (\sum_{i=1}^{n} m_i)}$.

F. Measuring performance

Given just the distance measure as noted above, it is hard to tell how much a network ends up learning and whether its performance is acceptable. Therefore, we use two other approaches as reference points for our LSTM networks. First, to determine what a network's minimal performance would be in the case it ends up learning nothing, we calculate the averages over all features used as output in the training set. Given these averages, we combine them with the known input features, determine the segments' positions accordingly and calculate the average Euclidean distance from the real segments' positions. Second, to determine the benefit of having a recurrency in the neural network, we compare the LSTM's average Euclidean distances with that of a standard FFNN using the same hyper-parameters (with exception of the number of timesteps used).

V. Results

A. Hyperparameters vs performance

See Figures 5 and 6 for two plots containing respectively the performances of LSTM networks using different input features and LSTM networks using different number of hidden units for each tested number of timesteps. Here the errors are obtained by comparing the estimated positions with the real positions over the whole testset, as explained in section IV-E. Looking at the figures, it can be seen that the averages as well as the standard deviations of networks using higher number of timesteps (i.e. 25 or 30) is slightly worse than those of the networks using fewer number of timesteps. These differences are marginal at best however, and generally speaking there does not seem to be much of a relation between the network's performance and the depth with which it propagates back through time. Next to that, in Figure 5 it can be seen that networks using only the orientations tend to outperform those that also use the angular velocity and angular acceleration. Similarly, in Figure 6 it can be seen that networks using 100 hidden units outperform similar networks using 30 hidden units.

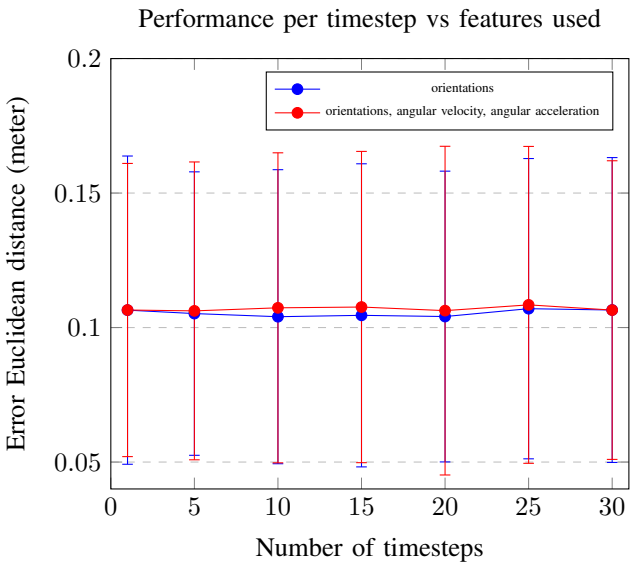


Fig. 5: Number of timesteps used vs performance for two kind of networks using different inputs. The blue line represents LSTM networks using only the orientations as input. The red line represents LSTM networks using the orientations as well as the angular velocity and angular acceleration as input.

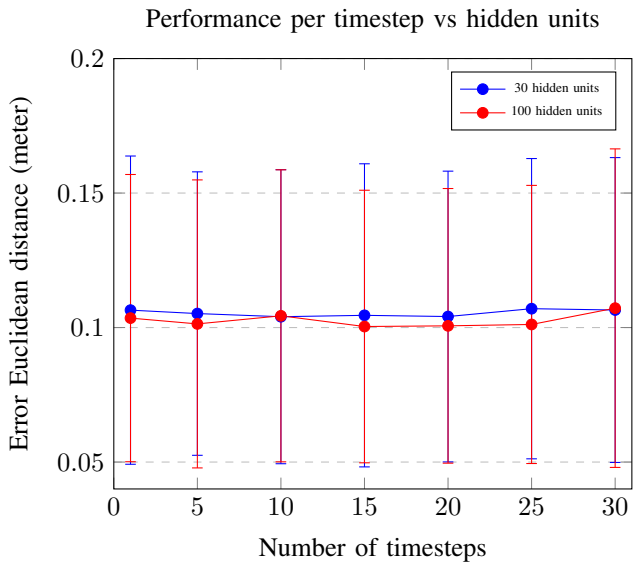


Fig. 6: Number of timesteps used vs performance for two kind of networks using different number of hidden units. The blue line represents LSTM networks using 30 hidden units. The red line represents LSTM networks using 100 hidden units. All networks in this graph use only the orientations as input.

B. LSTM vs FFNN vs average

From the networks of which the performances were seen in Figures 5 and 6, the best performing network (albeit marginally) was found to be the LSTM with 100 hidden units, using the orientations as input and using a timestep of 15. We compared this network with a FFNN containing the same number of hidden units and calculated their Euclidean estimation errors per exercise in the testset. The results can be found in Figure 7 together with the estimation errors obtained by using the averages of the output features in the training set as estimation. As can be seen in the table, the FFNN performs slightly better than the LSTM, despite being a much simpler network and having far fewer weights (recall from section II-C that a single memory cell in a LSTM is a composition of multiple units/weights). Next to that, it can be seen that both networks consistently perform better than the estimations that are based on taking the averages over the features in the training set. Although this indicates that the networks do end up learning some relations between in- and outputs, the differences are not substantial either for many exercises.

C. Visualizing exercises

Since the results in Figure 7 do not tell much about the synthesized motions in itself, we plotted and visually examined the exercises to determine the cause of the estimation errors. In this section, we present several plots from the series of exercises which depict the causes for the estimation errors the best. First, see Figure/video 8b for a plot of exercise 1, executed by the test person and synthesized by the FFNN using only the orientations as input (this network is the same as the FFNN of which the results were seen in Figure 7). When comparing this video with the original seen in 8a, it can be seen that the motion as synthesized by the FFNN contains little to no movement in the upper arms. In order to see whether this problem persists throughout other exercises and to determine the degree of this problem, we examined exercise 14, in which the participant was asked to rotate his/her arms around the shoulder and to touch his/her legs among others. See Figure/video 9b for a plot of the motion as synthesized by the FFNN using only the orientations as input. Looking at the video, it can be seen that the same problem persists, despite the fact that the orientations of the forearms cover a wide range of possible values in this exercise. Next to that, upon examination it also becomes apparent that the FFNN using only the orientations as input has severe problems estimating the spine. This is most visible when the test person touches his/her toes. While this motion causes the spine to bend in the original video as seen in 9a, the spine remains straight in the motion synthesized by the FFNN using the orientations as input. This same problem is also visible to a lesser degree in exercise 6, which can be seen in Figure/video 10b. As seen in Figure 7, this exercise was the one with the worst performance. Not only does the synthesized motion suffer from the same problems seen before (stagnant upper arms and spine), it also faces the problem of incorrectly estimated legs. When comparing the original motion as seen in 10a with the synthesized motion seen in 10b, it can be seen that the legs remain relatively stable in the former video whereas they move during the exercise in the latter video.

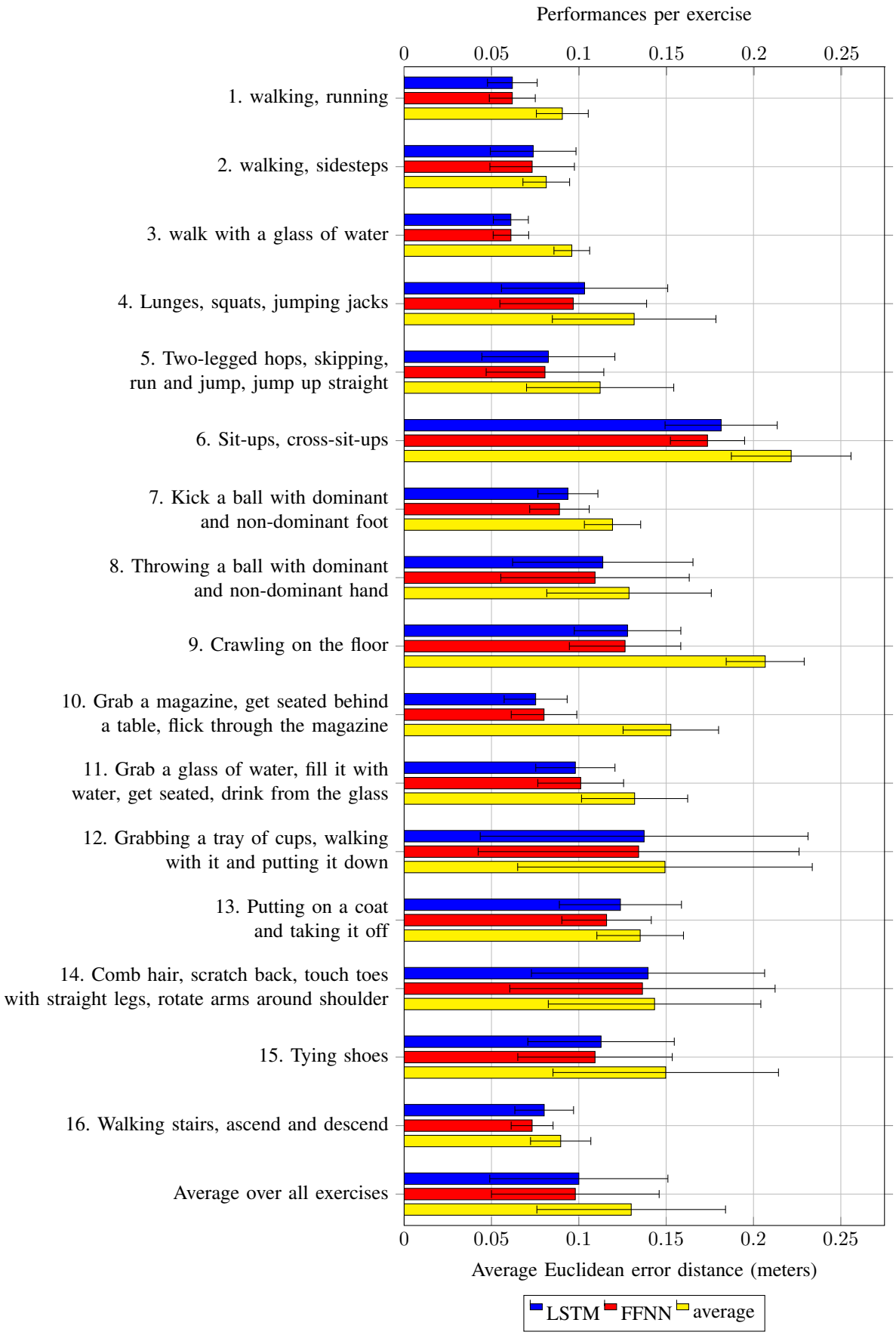
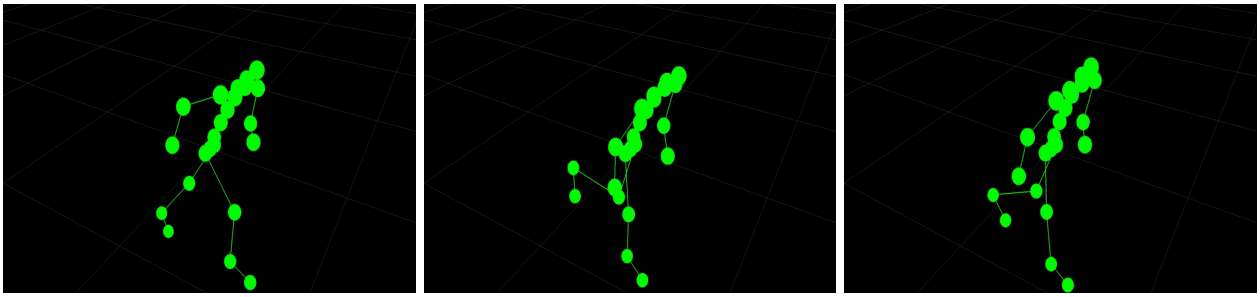


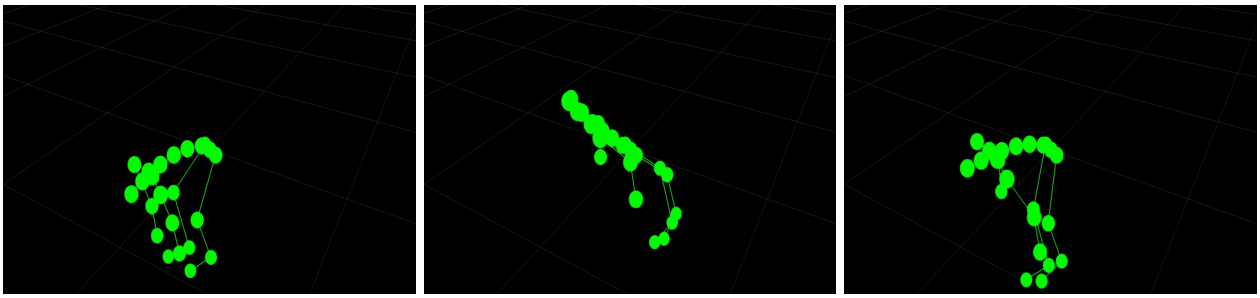
Fig. 7: Performances per exercise. Here 'average' (yellow) denotes the performances obtained by using static values as 'estimations', calculated by taking the averages over the output segments in the training set.



(a) The original (b) FFNN, orientations (c) FFNN, orientations+positions

Play all

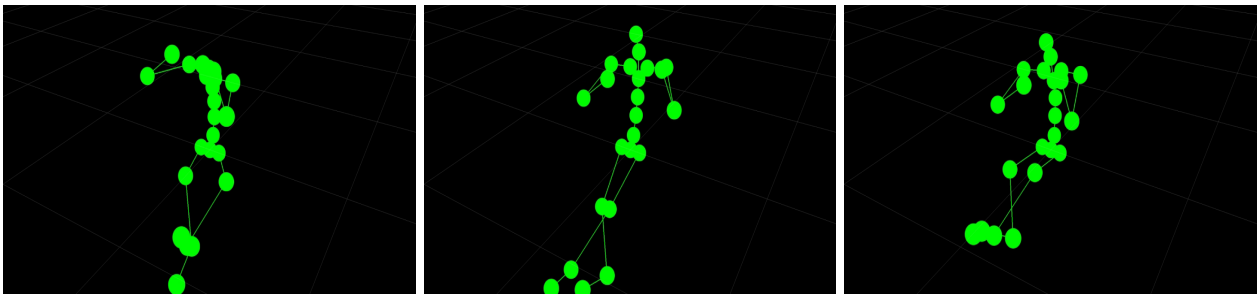
Fig. 8: Exercise 1: walking and running.



(a) The original (b) FFNN, orientations (c) FFNN, orientations+positions

Play all

Fig. 9: Exercise 14: Combing hair, scratching back, touching toes with straight legs, rotate arms around shoulder.



(a) The original (b) FFNN, orientations (c) FFNN, orientations+positions

Play all

Fig. 10: Exercise 6: sit-ups and cross-sit-ups.

D. Inclusion vs exclusion of positional data

Considering the problems that the FFNN faces when using only orientations as input, we tested whether these same problems remain when including positional data. See the Figures/videos in 8c, 9c and 10c for the motions as synthesized by a FFNN that includes positional data in addition to the orientations as input. Other hyperparameters of this network are kept the same as the FFNN that only used orientations as input. Looking at the videos, it can be seen that the inclusion of positional data greatly improves the network's capability of estimating the orientations of the spine and the upper arms. This is especially well visible in video 9c where the synthesized motion approximates the original much closer than the motion seen in 9b. Consequently, the FFNN that includes positional data as input has a much lower reconstruction error. While the FFNN using only orientations as input scored an average estimation error of $\mu = 0.098$ with a standard deviation of $\sigma = 0.048$ over the whole training dataset (as seen in Figure 7), the FFNN that includes positional data scored an average of $\mu = 0.079$ with a standard deviation of $\sigma = 0.034$. Although this change improves the performances, it must be noted however that the problem of legs moving remains in exercise 6 as seen in video 10c.

VI. Discussion

In the last section it was seen that the inclusion of the angular velocity and angular acceleration as input does not improve a network's performance. Furthermore, it was seen

that a network's performance is greatly improved by the inclusion of positional data in the input. When excluding positional data, it was found that both the LSTM and the FFNN were unable to correctly estimate the orientations of the upper arms and the spine among others. These differences in performance are the result of there being no clear one-to-one mapping from in- to output when only regarding the orientations as input. This is an undesired consequence caused by the fact that different body poses can generate the same orientations in the set of input segments. This can easily be seen when considering a single arm for example. It is possible to move the upper arm without changing the orientation of the forearm. As a result, the network simply learns the average over all outputs seen for a given input, causing it to output a (near) constant value. On the other hand, when positional data is included in the input, the number of possible outputs for a given input is far smaller.

Aside from that, it was seen that the performances of the LSTMs were roughly the same regardless of the number of timesteps used. Furthermore, results in Figure 7 have shown that the LSTM's performance was slightly worse than that of a FFNN. These results indicate that that the problem does not benefit from a recurrency with respect to time. This may partly be caused by the previously noted problem of having no one-to-one mapping from in- to output when using only the orientations as input. To elaborate this further, recall from Equation 3 that the activation of a memory cell in the LSTM is a combination of the input at the current timestep and the previous timesteps. Considering that no apparent relation may exist between in- and output when using only the orientations, the LSTM may learn that previous timesteps have no relation with the output. Consequently, the forget gate in the LSTM may learn to drop (almost) all information from previous timesteps, in which case the recurrency fails to provide any benefits. Although this may partly explain why the LSTM fails to outperform the FFNN, first preliminary tests using an LSTM that includes positions in addition to orientations as input have shown that this network also performs worse than a FFNN using the same features. Therefore, the lesser performance of the LSTM may also simply be caused by the difficulty of the problem. As we are dealing with a problem where a regression is performed on an output with a dimension of 72, finding a relation between inputs with respect to time and linking them to the consequences on the output may be an incredibly hard task.

Next to this problem, another issue that was seen in Figure/video 10 was the problem that the legs move in the synthesized motion whereas they remain relatively stable in the original. Likely, this is the result of the normalization approach that was taken. With the current normalization approach, all segments are normalized according to the orientation of the pelvis. As an undesired consequence, when the pelvis moves, all features change accordingly. Consequently, a change in the orientation of the pelvis may cause other segments to move in the motion as synthesized by the network. This is likely what happened in Figure/video 10b and 10c, since the pelvis moves during sit-ups. Another undesired effect of the normalization approach is that the networks are unable to differentiate between for example a person standing straight up or lying on the floor. Although this approach reduces the amount of possible in- and outputs, it also causes the network to lose information that may have been important for a correct estimation.

Summing up the problems discussed above, when omitting positional data, it was found that the networks were unable to estimate body poses correctly, regardless whether a LSTM or a FFNN was used. Comparing these performances with the performances reported in works such as that of Chai and Hodgins [1] and Liu et al. [5], who report results that are comparable to commercial mocap systems, we have found our approach to be of a lesser quality. This difference in performance can mainly be attributed to the lack of positional data however, since we have been able to estimate motion reasonably well with the inclusion of positional data. However, since accurately capturing this information using an inertial mocap system is problematic, it is discouraging to utilize this information. Lastly, it must also be noted that the works of Chai and Hodgins [1] and Liu et al. [5] use a motion database containing fewer exercises than ours, which makes the task of estimation motion inherently easier.

VII. Conclusion

We have presented an approach that uses a LSTM network for motion estimation using a small number of sensors and compared its performance with a standard FFNN. Our findings indicate that the problem does not benefit from a recurrency. Furthermore, it was found that features containing information with respect to time, such as the angular velocity and angular acceleration, to be unhelpful as input. Also, it was found that the exclusion of positional data in the input greatly decreases performance.

VIII. Future work

Although it was found that the LSTM was unable to outperform a FFNN and that the lack of positional data hinders a network's capability in finding a mapping from in- to output, several adjustments may be made that could help improve a network's performance. First, while concrete positions may not be obtained reliably from an inertial mocap system, it is expected that having general knowledge of how sensors are (roughly) located in space relative to each other is sufficient. Capturing such spatial information may be easier than capturing concrete positions, easing the problem.

Another change which could increase a network's performance is by (partly) including the orientation of the pelvis. Currently, all information about the pelvis and about a person's general orientation in space is lost. As a result, when the pelvis moves during a motion, the orientations of other segments may unintentionally change with it, as was seen in Figure/video 10. Giving the network the ability to distinguish between such body poses by including features from the pelvis may increase its performance.

Other improvements that can be made include adding a 6th sensor to the system. Currently, the segments located at the forearms and lower legs are used as input to the network,

while using the pelvis for normalization. The problem with this approach is that these segments provide very little information about the spine for example. To alleviate this problem, an extra sensor may be added. We propose to position this extra sensor at e.g. the T8 or the neck such that the orientation of the spine is included in the input. Our first preliminary result using a FFNN that includes the T8 while using positions and orientations as input, achieved an average Euclidean estimation error of $\mu = 0.067$ and a standard deviation of $\sigma = 0.029$ over the training dataset. This is a reasonable improvement compared to the $\mu = 0.079$ and $\sigma = 0.034$ when using only the original 5 segments.

Finally, the performance could possibly also be increased by distributing the task of estimating orientations over a cluster of multiple neural networks. Currently, a fully connected network that maps all input segments to all output segments is used. As a consequence, the network tries to find relations between e.g. the lower legs and the upper arms, whereas they are expected to have little relation with each other. To solve this, we propose to use a cluster of 5 networks; one for each arm, one for each leg and one for the torso, each one using only input segments that are expected to have a strong relation with the output. Once trained, these networks could be combined to form a single, sparsely connected neural network.

All in all, a wide range of different approaches remain that have yet to be investigated more closely. It is our belief that much better results can be obtained using the right combination of approaches. This belief is based under the assumption that proper spatial data can be obtained from a system with few sensors, as spatial data is believed to be invaluable.

ACKNOWLEDGMENTS

I would like to thank Mannes Poel and Frank Wouda for their support during the project and for their review on this paper. Furthermore, I would like to thank Frank for providing the dataset which ultimately made this project possible.

REFERENCES

[1] Jinxiang Chai and Jessica K Hodgins. Performance animation from low-dimensional control signals. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 686–696. ACM, 2005.

[2] Terence David Sanger. Human arm movements described by a low-dimensional superposition of principal components. *The Journal of Neuroscience*, 20(3):1066–1072, 2000.

[3] Nikolaus F Troje. Decomposing biological motion: A framework for analysis and synthesis of human gait patterns. *Journal of vision*, 2(5):2–2, 2002.

[4] Guodong Liu, Jingdan Zhang, Wei Wang, and Leonard McMillan. Human motion estimation from a reduced marker set. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 35–42. ACM, 2006.

[5] Huajun Liu, Xiaolin Wei, Jinxiang Chai, Inwoo Ha, and Taehyun Rhee. Realtime human motion control with a small number of inertial sensors. In *Symposium on Interactive 3D Graphics and Games*, pages 133–140. ACM, 2011.

[6] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[7] Pedro Nogueira. Motion capture fundamentals. *Faculdade de Engenharia da Universidade do Porto*, 2011.

[8] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[9] Daniel Roetenberg, Henk Luinge, and Per Slycke. Xsens mvn: full 6dof human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV, Tech. Rep*, 2009.

[10] Alex Graves. *Supervised sequence labelling*. Springer, 2012.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[14] Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. Motion reconstruction using sparse accelerometer data. *ACM Transactions on Graphics (TOG)*, 30(3):18, 2011.

[15] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4346–4354, 2015.

[16] Sebastian Bitzer and Stefan J Kiebel. Recognizing recurrent neural networks (rrnn): Bayesian inference for recurrent neural networks. *Biological cybernetics*, 106 (4-5):201–217, 2012.

[17] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[18] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[20] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[21] Tom Schaul, Ioannis Antonoglou, and David Silver. Unit tests for stochastic optimization. *arXiv preprint arXiv:1312.6055*, 2013.